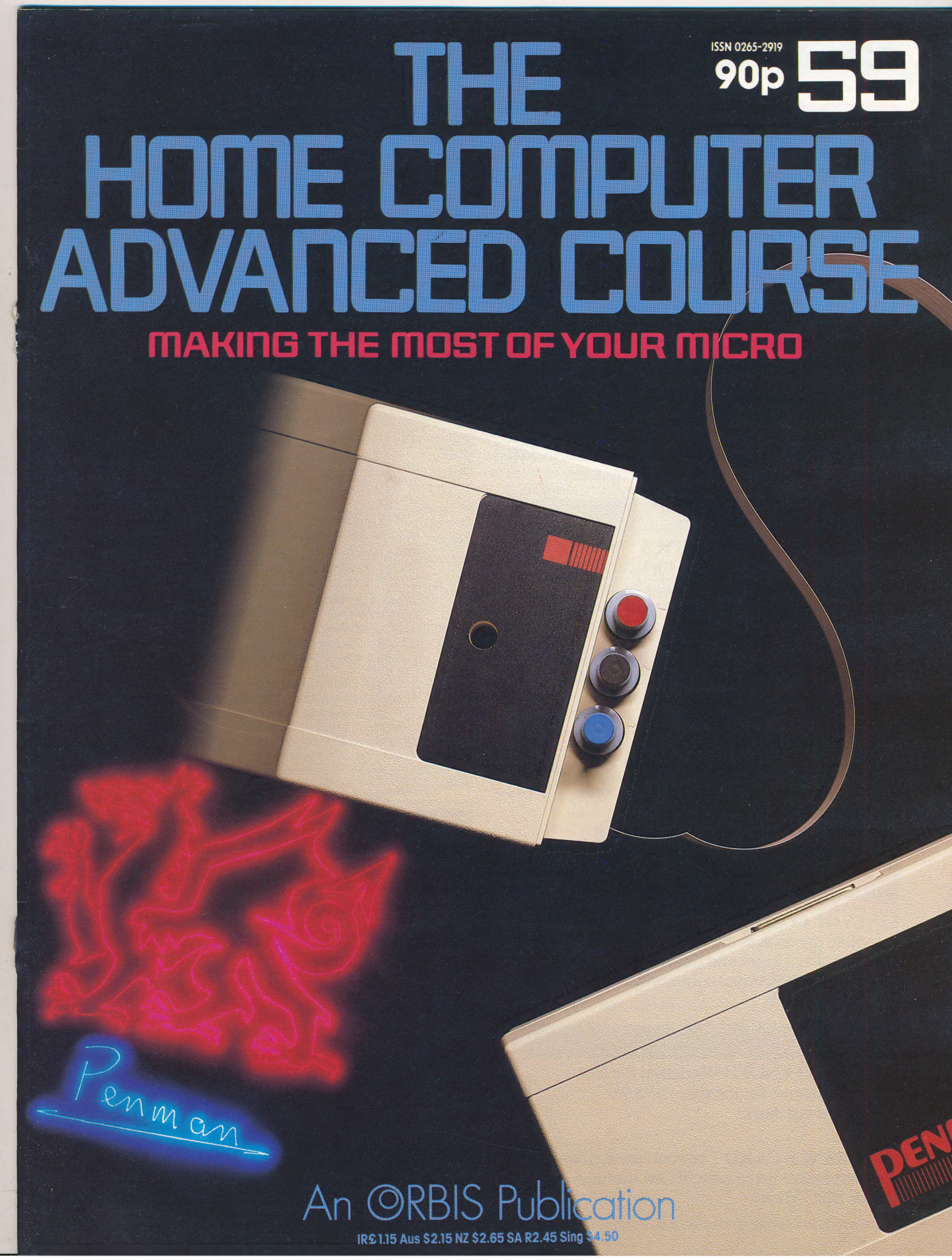
ISSN 0265-2919

90p 59

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

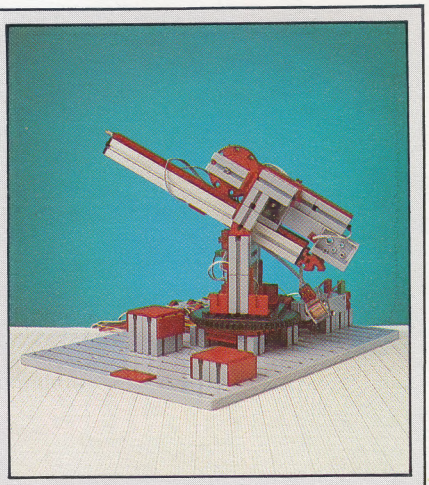
CONTENTS

Next Week

● We take a look at the Fischertechnik robotics kit and its interface, which allow a number of different computer-controlled robots to be constructed.

● In the concluding instalment of our present series on computers in education, we take a look at what future developments are on the way.

● As our simulation game trading ship approaches the New World, we begin to program the major events that can be introduced into the scenario.



QUIZ

1) Why does the Penman Plotter require a central processing unit on board?
2) What advantage, in terms of the memory size required, does a hierarchical database manager have over a relational structure?
3) Why would we use a 'parameter list'?

Answers To Last Week's Quiz

1) The 'noise margin' is the tolerance of an electronic device before it generates errors. Therefore, a noise margin should be as high as possible.
2) 'Packing' enables information to be compressed within the array, thus conserving memory.
3) The Apple II runs on a 6502 processor and this would not normally be able to make use of dBase II. However, the addition of a Z80 second processor makes it possible to run the program.
4) The Commodore 64's serial port differs from the RS232 standard by using different ASCII codes and having a 5v power supply instead of the normal 12v.

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Deputy Editor** Steve Colwill; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Steve Malone; **Sub Editor** Jon Kaye; **Contributors** Dr Andrew Bangham, Nick Walsh, Dr Antonia Jones, Steve Malone, David Mudd, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY PAUL CHAVE

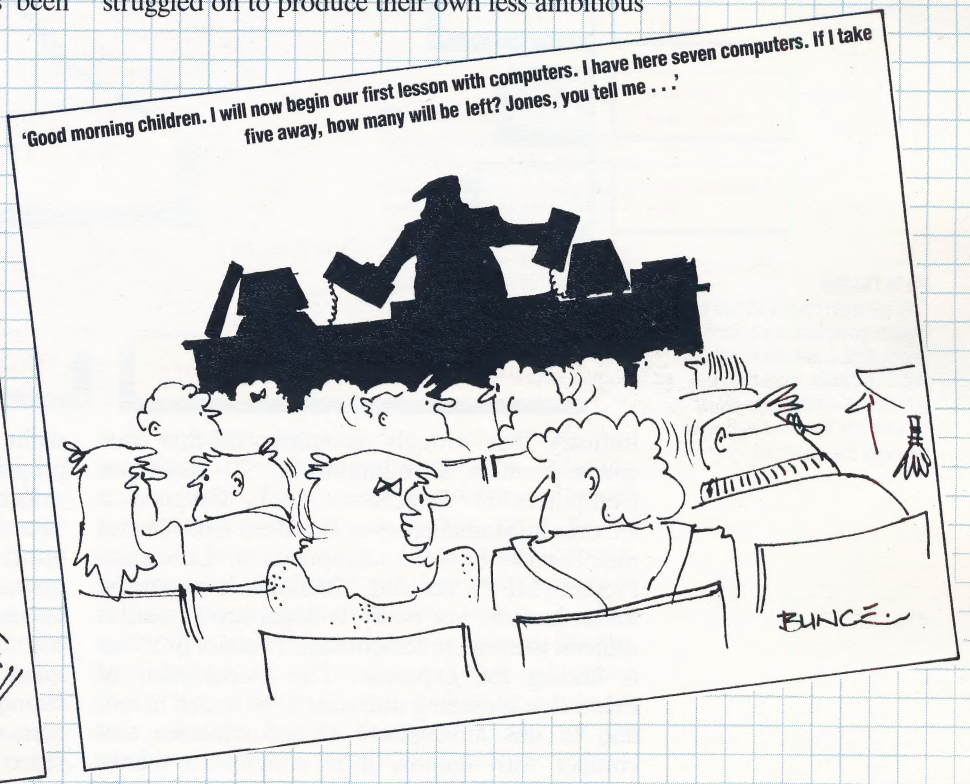
A LESSON IN ECONOMICS

The introduction of computers into education has brought with it a whole host of problems – not least of which is the lack of government funding. We look at the range of difficulties confronting teachers grappling with new technology, from the need for increased security to confronting sexual discrimination.

The introduction of computers into education is not taking place without some difficulty. Economic cuts, widespread ignorance about the subject and difficulties in educating both trainee and in-service teachers all combine to place severe restrictions on educational computing. New technology is suffering from recent government policies and the resulting drastic cuts in public sector spending are creating a scarcity of these resources in schools. Having bought expensive hardware, many teachers are angered by school budgets that do not allow for buying the software to run the machines.

Equally, this lack of funding has prevented the setting up of large research projects involving computers and children. Little thought has been given to how computers affect an existing syllabus and, consequently, no clear strategy has been developed for introducing them into schools. TICCIT and PLATO in the US and the National Development Program in Britain (see page 1001) were both set up in the 'pre-micro' years of the early 1970s when finance was far more readily available. Now, in the fiscally squeezed 1980s, governments in both countries cannot find the resources to fund similar projects when they are desperately needed. Other problems include the fact that no serious studies have been made of the place of computers in the curriculum. In secondary schools, the emphasis on BASIC and the existence of Computer Studies as an academic discipline in its own right are under attack. Primary schools are given a computer and told to 'get on with it' and teachers must muddle through as best they can.

There are sometimes serious breakdowns of communication within the educational computing world. In London, many schools have invested in RML machines, and, consequently, ILECC — the Inner London Educational Computing Centre — spent a lot of time and effort trying to write a version of LOGO for the RML micros. Meanwhile, Research Machines Limited itself produced a respectable, full implementation of the language. ILECC, throwing good money after bad, then struggled on to produce their own less ambitious



turtle graphics program, and are in the position of having to promote this to schools when there is a superior product on the market.

Further problems have arisen with the absence of a clear policy for training teachers. Computers have been deposited in classrooms and teachers have been expected to learn everything from the manual. Financial restrictions have inhibited the establishment of in-service courses and when they have been available, schools have been reluctant to send their teachers' because of the disruption their absences cause.

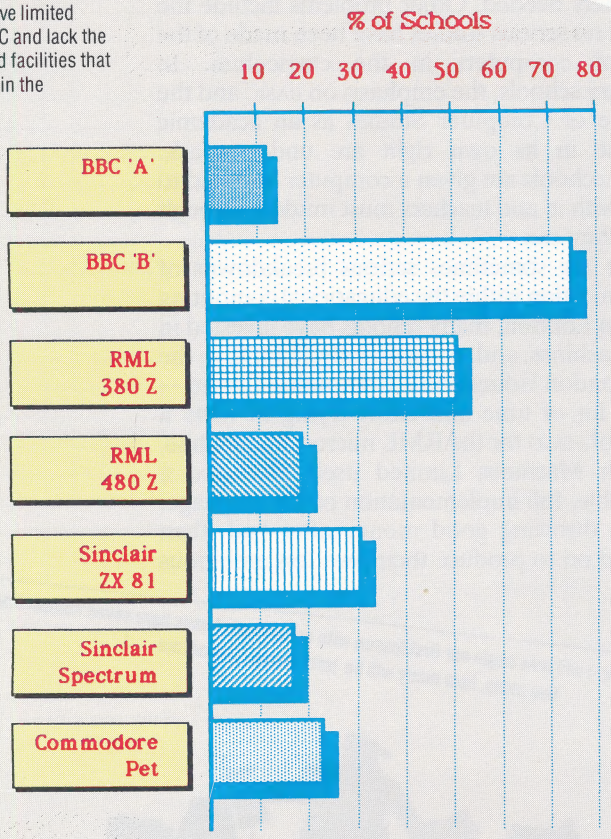
The situation in teachers colleges is even worse. In 1983, the Department of Education was worried about the lack of computer courses for trainee teachers. Supported by the Department of Industry, they strongly recommended that each course contain a minimum of 20 hours on Computers in Education. Chris Gregory, a lecturer in Mathematics at Bradford College and member of the Microcomputers in Education Project (MEP), has said: '20 hours is a pathetic token gesture, yet many colleges are finding it difficult to even implement that.' A major problem is finding the expertise. The Department of Education is causing staff sizes to be frozen or cut, and in this atmosphere of redundancies and conflict with unions, it is difficult to recruit computer staff.

Good quality educational programs have been slow in appearing. Governments have spent money on hardware but not software, and since schools have no money to spend, the educational software market is in the doldrums. At the Educational Publisher's annual software exhibition in 1983, it was revealed that the majority of software houses in this field were losing money and many were expected to go out of business in the following year. What depresses the market further for educational publishers is that many of the larger education authorities have invested money in developing their own software, which is distributed at low cost, or free, to schools. This, of course, make the more expensive commercial programs less appealing.

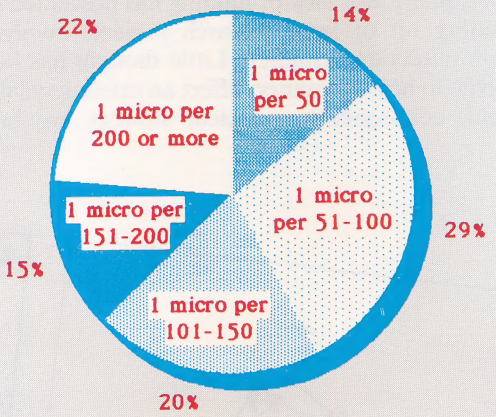
For example, Sloane Software recently released a maths program called Hopslide, for five to seven year olds. It is a simple program, designed to generate an understanding of the value and order of numbers. Children are presented with a row of numbers between five and nine. They 'hop' them over each other, or 'slide' them into a space. Hopslide is modestly priced, at £6.50, and could be expected to sell quite well. The Inner London Education Authority produce their own software. A program called Hopslide also appears in a maths package, produced by their SMILE mathematics centre. It is the same game as the Sloane Software version, except the range of numbers is from two to nine, and the graphics are slightly better. The SMILE pack contains 19 other programs of similar quality and costs only £8.00.

Lack of finance and poor organisation are, of course, not confined to educational computing, but there are some other, less obvious, problems that arise when trying to introduce new technology into schools. For example, the physical presence of computers in schools has caused problems. In many schools, electronic equipment is stored in a strongroom overnight. Before the arrival of computers, this involved two or three cassette and video recorders. In one primary school with a micro in every classroom, the teachers had to make two or three journeys with a computer,

Teachers' Pets
Endorsed by the BBC and recommended by the government for use in schools, Acorn's BBC machines have secured a firm foothold in the British educational system. The figures shown on these pages are all drawn from a government-funded survey published in January 1985 by the BBC Educational Broadcasting Services Research Unit. One surprising feature of the results is the continued presence of the old Commodore Pet series and the Sinclair ZX81, both of which have limited versions of BASIC and lack the colour and sound facilities that can be so useful in the classroom



Types of Micros in Schools



Pupil/Micro Ratio

GRAPHS BY IAN McKINNEL ON THE MACINTOSH

Pie In The Sky
This pie chart clearly shows that despite government attempts to improve the pupil/micro ratio in British schools, more than one school in five still has to make do with less than one computer for every 200 children!

monitor, leads, disk drive and software down a long corridor and up two flights of stairs, every morning and evening. After two accidents, the teachers refused to carry the equipment and the education authority concerned refused to sanction leaving the computers in the classroom overnight. The result was that all computers were left in the strongroom for four months while the teacher's union and the education authority negotiated a compromise. The conflict was finally resolved by installing steel storage cabinets near each classroom.

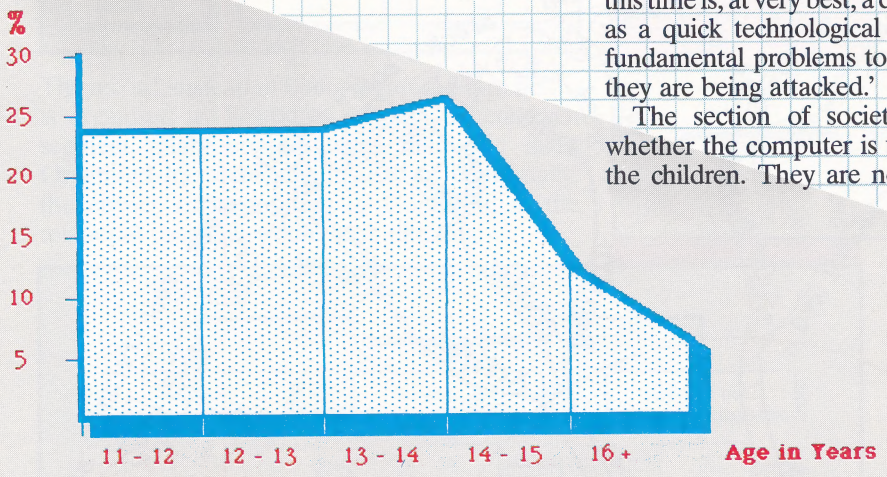
The security situation in many schools is deplorable, for the simple reason that education authorities have no money to invest in reasonable security systems. Most inner-city schools in Britain are burgled or vandalised several times a year. The insurance premium is so high that many authorities find it cheaper to replace stolen goods than insure them.

Other problems include sexual discrimination, particularly since computers in school and the home tend to appeal to boys. Most home computers are owned by teenage males who tend to use them for playing male-oriented games that are often violent and militaristic. This attitude has spilled into the classroom — the computer is often regarded as a male preserve. Many educational programs are male-oriented and boys often monopolise the computer at the expense of the girls. A spokesperson for the Equal Opportunities Unit of the Inner London Education Authority said, 'We are aware of the male bias in Computer Studies. In 'all girls' schools they get on happily with the subject. It's only a problem in mixed schools, where the boys tend to be dominant. In mixed schools, we find we have to actively encourage the girls to take an interest in computers to redress the balance.' One of the interesting results to come out of research at Edinburgh University, using LOGO in schools, was that it appealed to girls as much as boys.

Another major difficulty is that governments continue to see education in terms of productivity and cost. Looking at the computer as a way of improving educational productivity implies a frightening future, with children becoming part of a computerised production line, teachers replaced by machines and curriculums and lessons determined by government committees. As Seymour Papert remarked: 'There are tremendous forces in the world pushing towards a more rigid society, towards using the computer in a way which makes life less exciting, more rigid and less social. Those forces are powerful. What can we do in the face of them? . . . We can say "Burn the Computers". That is a waste of time . . . Or we can try to push as hard as we can in the opposite direction and use the computer in ways which will go in a socially desirable direction.'

Education on both sides of the Atlantic is in crisis. Lack of investment, low wages and morale, and a general deterioration of social conditions are all taking their toll. There are those who argue that introducing computers into schools distracts from the real problems threatening education. Joseph Wiezenbaum, Professor of Computer Science at the Massachussets Institute of Technology said recently: 'Our schools are already in desperate trouble, and the introduction of the computer at this time is, at very best, a diversion . . . It is used . . . as a quick technological fix . . . to paper over fundamental problems to create the illusion that they are being attacked.'

The section of society that finally decides whether the computer is used or abused may be the children. They are never shy about voicing their opinions and they are aware of whether something is beneficial and stimulating, or boring and irrelevant. They recognise and respond to well thought-out software, and often have to be dragged away from it. In a computer-rich environment, children create their own 'computer culture,' developing their own jargon, helping each other with programming problems, sharing new discoveries, exchanging software, educating themselves collectively and evaluating hardware and software. It may be the children, not the teachers, advisers, committees and researchers, who finally decide how computers can be used most beneficially in education.



Who Misses Out?
This graph, representing 'hands on' computer experience among schoolchildren in Britain, clearly shows the effects of specialisation at the upper end of the age range, where older students may opt for subjects that make no use of computers. More disturbing, however, is the fact that none of the age groups claim a 'hands on' rating higher than 30 per cent

'Hands on' Experience

GOOD RELATIONS

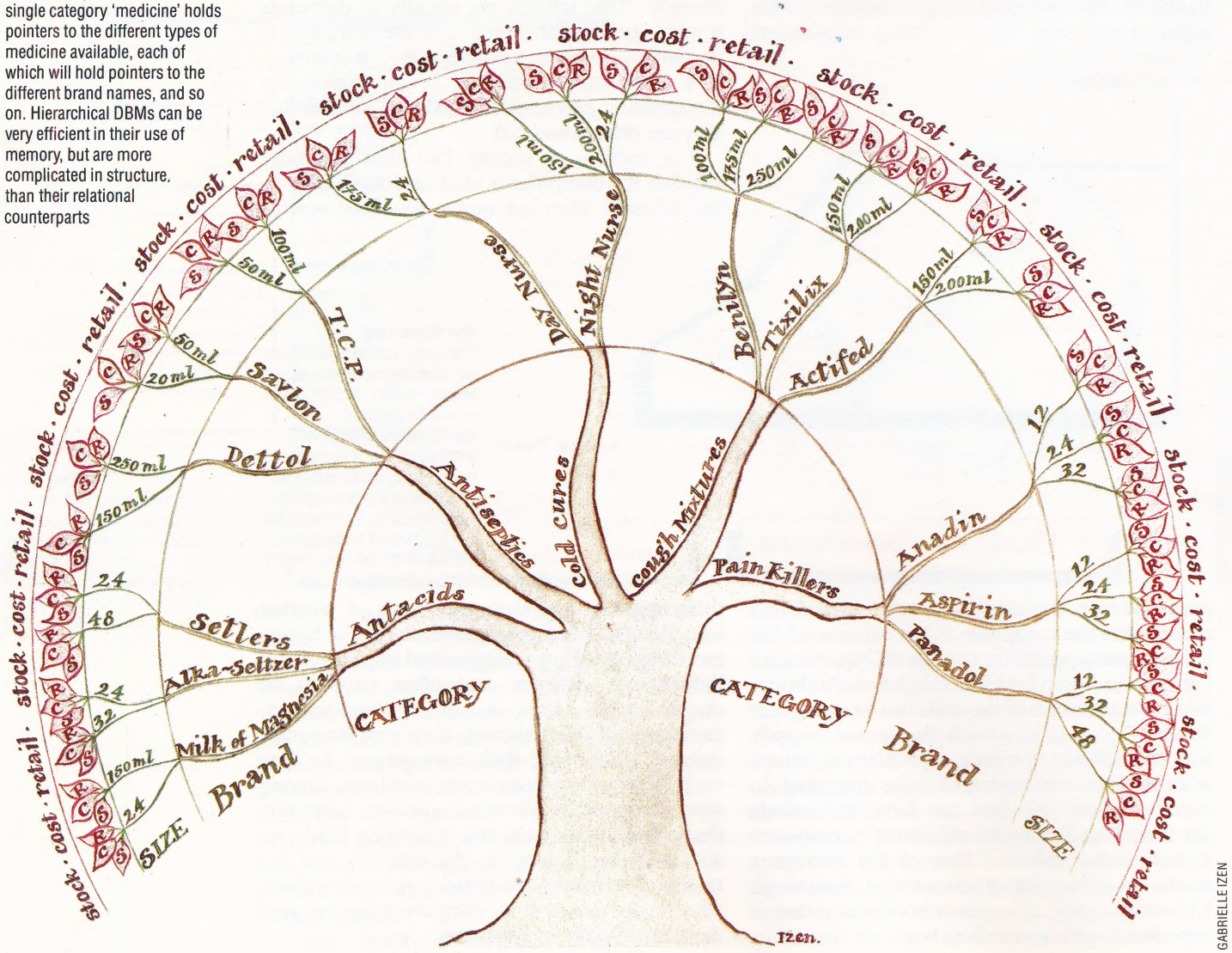
Database managers implemented on home micros generally incorporate 'relational organisation' methods as a means of arranging the input data. We look at the structure and functioning of this type of system, as well as its two alternatives – hierarchical and network systems.

Most of the DBMs available on home computers are of the type known as 'relational' database management systems. At its simplest, this means that each record in a DBM file takes the form of a table, made up of rows and columns, after the fashion of a spreadsheet. The various ways in which the information can be presented might suggest a more complex structure, but it is basically nothing more than a straightforward table or grid. The database is organised in much the same way as you might yourself put the information down on paper. Each record is structurally the same and each field within the record is of a fixed length.

The term 'relational' comes from the fact that each 'row' in the database is clearly related, in a fixed and rigid way, to each 'column'. This is not a very flexible way of organising data, but it does make for relatively simple database manager programs. Because home computers — even the new 16-bit machines with 128 Kbytes of memory or more — have comparatively small memories, slow processing speeds, and limited data storage capacities, the restrictions of a relational system are part of the price that must be paid for combining affordable computing with database management.

An altogether different approach to this tabular organisation of data is to order it in a hierarchical form. We can think of the data being organised as if it were a tree, with each branch having sub-branches, the sub-branches having twigs, even leaves. To illustrate this, we'll create a database of the stock in a newsagent's shop. First, we'll present it in a relational way, then show how it would be

Tree Surgery
Whereas a 'relational' DBM stores information in tabular form, a 'hierarchical' DBM organises its data in 'tree' form. In the example shown, the single category 'medicine' holds pointers to the different types of medicine available, each of which will hold pointers to the different brand names, and so on. Hierarchical DBMs can be very efficient in their use of memory, but are more complicated in structure, than their relational counterparts

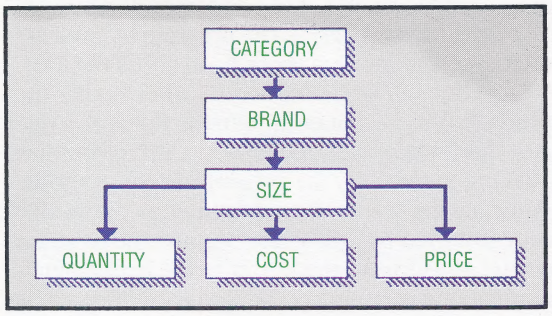


GABRIELLE IZEN

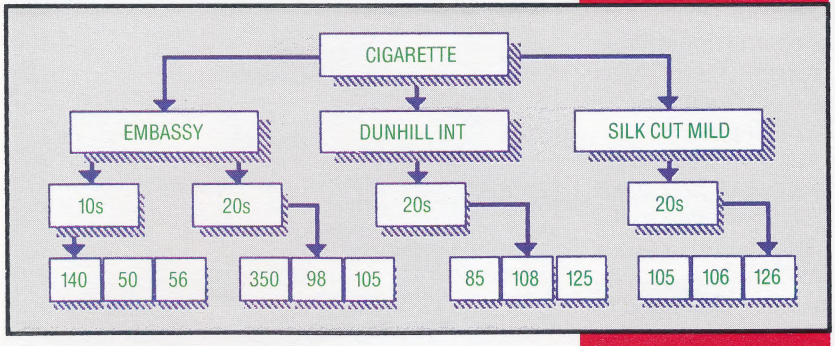
organised hierarchically.

CATEGORY	BRAND	SIZE	STOCK	COST	RETAIL
CIGARETTE	EMBASSY	20	350	98	105
CIGARETTE	EMBASSY	10	140	50	56
CIGARETTE	DUNHILL INT	20	85	108	125
CIGARETTE	SILK CUT MILD	20	105	106	126
SWEETS	BOUNTY	1	106	14	17
SWEETS	TWIX	1	95	12	16
SWEETS	MARATHON	1	25	15	19
MAGAZINE	YOUR SPECTRUM	1	35	85	95
MAGAZINE	NEW STATESMAN	1	12	60	80
MAGAZINE	CITY LIMITS	1	86	50	60
DRINKS	LUCOZADE	150	35	25	37
DRINKS	TIZER	150	40	18	27
DRINKS	QUOSH	150	20	16	25

When represented hierarchically, this newsagent's stock database could look like this:



The record, instead of being organised into fields, is organised as 'segments' of data, where each segment might contain more than one field. Organised hierarchically, instead of relationally, the entries corresponding to CIGARETTE in the relational database would look like this:

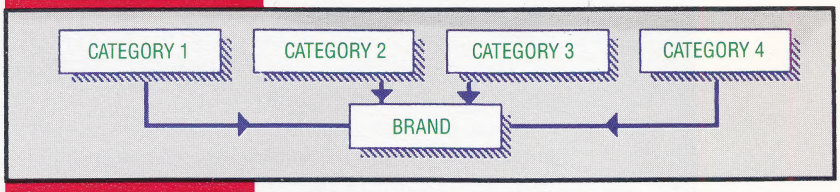


When it is represented in this way, each item of information can clearly be seen as being either superior or subordinate to another item of data. All that is required is a 'pointer' from one data segment to another. In this example, we need only one CIGARETTE segment, with pointers to the various types of cigarette held in stock, and further pointers to the sizes, cost, price and so on.

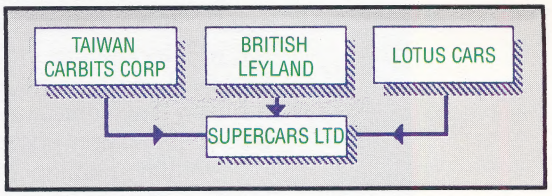
Suppose the newsagent has just one hundred 'categories' of stock, but a thousand lines on sale; with a relational database, one thousand records would be required — one for each line. With a hierarchical database, only one hundred segments would be needed to cover the entire stock. It saves a lot of duplication but makes the database much more complicated structurally.

There is a further type of database organisation known as the CODASYL or network system, a standard database organisation specified by the database task group of the Conference on Data Systems Languages — hence CODASYL. The problem with a hierarchical database is that the data can only be organised to flow in one 'direction'; to use the tree analogy, one branch cannot be attached to two trunks. But, to return to our newsagent, EMBASSY PANATELLAS, for example, could be a 'branch' of both EMBASSY and CIGARS. The problem particularly comes to the fore whenever a single 'component' can have more than one 'maker'. If you sell auto spares, a single 'big end linking gasket' for a BL Marina could be made by British Leyland, BL Bargain Parts Ltd, or Taiwan Carbits Corporation. As we can see, the relationship between goods supplied and parts suppliers can become very complicated.

In the CODASYL system, a network of sets is used in which each set consists of a collection of records. Unlike a record in a relational database system, the record length need not be fixed, and any record can belong to more than one set. The CODASYL system allows a set to consist of only one record, and a record cannot belong to more than one occurrence of the same set type. Structures of the following type are therefore not allowed:



This is a serious limitation. Although we might not have CIGARETTEs, MAGAZINEs and DRINKs all with a brand name of, say, FABULAN, it is easy to conceive of a situation where a company called Supercars Ltd. could be part of a structure like:



Although hierarchical and CODASYL databases are more flexible than relational ones, they are more complex and really need the power of mainframe or minicomputers. Home computers almost invariably use relational systems. The commonest problem to arise from this limitation is in situations where, for example, you have an inventory of parts (or books, or other goods) that you sell, and each part (book, etc.) has more than one supplier. Therefore, you need a record of parts, and a linked set of records for related suppliers for each part. Multi-file DBMs allow more than one file to be open at a time, and for an active file to refer to a subsidiary file.

LINES OF COMMUNICATION

In the previous instalment, we inspected each of the Commodore 64's I/O ports — right down to the function of the last pin connection — and looked at how the OS controlled RS232. Now we can consider how the two 6526 CIA chips control I/O, with specific reference to the program Parawedge, which is designed to send or receive a specified block of memory through the user port.

The Commodore 64 has two 6526 CIA (Complex Interface Adaptor) chips, which specifically handle communications with the outside world. These aren't the only chips with input and output capability, however; the 6510 and video chips both handle aspects of I/O as well. A 6526 chip has two eight-bit data ports, both of which have individually programmable lines. The chip is capable of eight- or 16-bit communication, and has two linkable 16-bit timers. In addition, it has an eight-bit shift register for serial communication, and, as we saw in the first article in this series (see page 1139), a 24-hour programmable 'time of day' clock.

It also has two specific handshaking lines — PC and Flag. PC will go low for one cycle after data is written to port B of the 6526, and can be used to indicate 'data ready' to an external device. The Flag line may be used as a control input from another device. This can be used to set the Flag bit in the interrupt register, and — if so enabled — cause an NMI interrupt to the 6510 processor.

On the Commodore 64, the two 6526 chips handle different aspects of I/O: one of these (CIA #1, with base address at $DC00) is devoted to the keyboard and the joysticks, and the other (CIA #2, with base address at $DD00) controls data on the serial and user ports. The video chip handles I/O to the monitor, and the cassette port is handled directly by the 6510.

The Parawedge program we give here clearly illustrates the steps necessary to program the 6526 directly for I/O. The routine is an NMI wedge that uses the Flag line. It is designed to send a specified block of memory out of the user port as parallel data, or, alternatively, receive eight-bit parallel data until a specified block of memory is full. Since this is wedge code (see page 1139) it will clock data in or out on NMIs, while leaving the machine free to carry out other tasks the rest of the time. The only snag is that if the data rate becomes too high the Commodore 64 will begin to spend all its time doing NMI service routines, and that could be extremely awkward.

Entering Parawedge
We include an Assembly code listing for Parawedge, which can be typed in and assembled using an assembler. Alternatively, the program can be entered as a series of DATA statements by typing in and running the BASIC loader code

Parawedge allows the Commodore 64 to set up eight-bit parallel two-way communications with an external device — such as another computer or a parallel printer — through the user port. The pins of the user port from PB0 to PB7 are used for data transfer; the Flag 2 pin is used as the handshaking input; PA2 signals a 'ready for data' condition and PC2 signals a 'data valid' condition. To use the program, you must first define an area of RAM from which you wish to send outgoing data, or into which you wish to receive incoming data. This is done by passing the start and end addresses (in lo-byte/hi-byte form) to the program by POKEing them into the four locations from 50768 to 50771.

Location 50772 is used to indicate whether the program is to output or input data. POKEing a one into this location sets the program up for output; POKEing a zero sets the program to input mode. After having set up these parameters the wedge code is started by SYS 50775. Because the program is interrupt-driven, it will run in the background, sending or receiving data, while a BASIC program is typed in or run in the foreground.

Commodore 64 Parawedge Program

BASIC Loader

```
1000 REM ** PARAWEDGE BASIC LOADER **
1010 DATA173,84,198,208,61,169,0,141,3
1020 DATA221,169,144,141,13,221,173,2
1030 DATA221,9,4,141,2,221,173,0,221,9
1040 DATA4,141,0,221,173,80,198,133,251
1050 DATA173,81,198,133,252,173,24,3
1060 DATA141,85,198,173,25,3,141,86,198
1070 DATA120,169,188,141,24,3,169,198
1080 DATA141,25,3,88,96,169,255,141,3
1090 DATA221,169,144,141,13,221,173,24
1100 DATA3,141,85,198,173,25,3,141,86
1110 DATA198,120,169,234,141,24,3,169
1120 DATA198,141,25,3,88,96,169,144,44
1130 DATA13,221,240,36,173,1,221,145
1140 DATA251,230,251,208,2,230,252,173
1150 DATA82,198,197,251,173,83,198,229
1160 DATA252,144,49,173,0,221,41,252
1170 DATA141,0,221,9,4,141,0,221,108,85
1180 DATA198,169,144,44,13,221,240,246
1190 DATA177,251,141,1,221,230,251,208
1200 DATA2,230,252,173,82,198,197,251
1210 DATA173,83,198,229,252,144,3,108
1220 DATA85,198,120,173,85,198,141,24,3
1230 DATA173,86,198,141,25,3,88,108,24
1240 DATA3
1250 DATA25596:REM*CHECKSUM*
1260 CC=0
1270 FORI=50775TO50971
1280 READX:CC=CC+X:POKEI,X
1290 NEXT
1300 READX:IFCC<>XTHENPRINT"CHECKSUM
  ERROR"
1310 END
```

Assembly Listing

```
;*************************************
;*                                   *
;*       PARAWEDGE - A SEND/RECEIVE  *
;*                                   *
;*  WEDGE PROGRAM FOR 8 BIT PARALLEL *
;*                                   *
;*    COMMUNICATIONS ON THE CBM 64   *
;*                                   *
;*************************************
;
  CIA2 = $DD00      ;6526 CHIP BASE ADDR
  OUTPUT = $FF
  INPUT = $00
  OUTSHK = $04
  INTMSK = $90
  TOGHI = $04
  TOGLO = $FC
  NMIVEC = $0318
  ZPTEMP = $FB
  * = $C650
  START *=*+2       ;START ADDRESS
  END *=*+2         ;END ADDRESS
  MODE *=*+1        ;INPUT/OUTPUT FLAG
  VECTOR *=*+2      ;STORAGE FOR NMI VECTOR

  LDA MODE          ;INPUT OR OUTPUT
  BNE OUTDAT        ;BRANCH IF OUTPUT
  LDA #INPUT
  STA CIA2+3        ;SET DDR FOR INPUT
  LDA #INTMSK
  STA CIA2+13       ;FLAG INTERRUPTS DISABLED
  LDA CIA2+2
  ORA #OUTSHK
  STA CIA2+2        ;SET PA2 FOR OUTPUT
  LDA CIA2
  ORA #TOGHI
  STA CIA2          ;SET HANDSHAKE LINE PA2 HIGH
  LDA START
  STA ZPTEMP
  LDA START+1       ;MOVE POINTERS TO ZERO PAGE 0
  STA ZPTEMP+1
 ;
 ;INITIALISE INPUT WEDGE
 ;
  LDA NMIVEC
  STA VECTOR        ;SAVE OLD NMI VECTOR
  LDA NMIVEC+1
  STA VECTOR+1
  SEI               ;
  LDA #<NXTIN       ;
  STA NMIVEC        ;INSERT DATA-INPUT WEDGE
  LDA #>NXTIN       ;
  STA NMIVEC+1      ;
  CLI               ;
  RTS
 ;
 ;INITIALISE OUTPUT WEDGE
 ;
OUTDAT
  LDA #OUTPUT
  STA CIA2+3        ;SET DDR FOR OUTPUT
  LDA #INTMSK
  STA CIA2+13       ;FLAG INTS DISABLED
 ;
 ;
  LDA NMIVEC
  STA VECTOR
  LDA NMIVEC+1      ;SAVE OLD NMI VECTOR
  STA VECTOR+1
  SEI               ;
  LDA #<NXTOUT      ;
  STA NMIVEC        ;INSERT DATA-OUTPUT WEDGE
  LDA #>NXTOUT      ;
  STA NMIVEC+1      ;
  CLI               ;
  RTS
 ;
 ;
 ;INPUT DATA SERVICE ROUTINE
 ;
 ;
NXTIN
  LDA #INTMSK       ;CHECK ICR
  BIT CIA2+13       ;INT CAUSED BY FLAG?
  BEQ NOTCOM        ;NO.. NORMAL NMI
 ;
 ;OK BYTE ON PORT
 ;
  LDA CIA2+1        ;READ BYTE
  STA (ZPTEMP),Y    ;STORE IN MEMORY
  INC ZPTEMP
  BNE TEST1         ;INCREMENT POINTER
  INC ZPTEMP+1
 ;
TEST1
  LDA END
  CMP ZPTEMP
  LDA END+1         ;CHECK TO SEE IF ENDED
  SBC ZPTEMP+1
  BCC DONE          ;BRANCH IF FINISHED
 ;
 ;TELL DEVICE READY FOR NEXT BYTE
 ;
  LDA CIA2
  AND #TOGLO
  STA CIA2          ;TOGGLE PA2 LOW THEN HIGH
  ORA #TOGHI
  STA CIA2
 ;
 ;NOW DO NORMAL NMI ROUTINE
 ;
NOTCOM
  JMP (VECTOR)


 ;
 ;
 ;OUTPUT DATA SERVICE ROUTINE
 ;
 ;
NXTOUT
  LDA #INTMSK       ;CHECK ICR
  BIT CIA2+13       ;INTERRUPT CAUSED BY FLAG?
  BEQ NOTCOM        ;NO.. DO NORMAL NMI
 ;
 ;OK SEND BYTE
 ;
  LDA (ZPTEMP),Y    ;GET BYTE FROM MEMORY
  STA CIA2+1        ;OUTPUT IT. PC WILL GO
                    ;LOW FOR 1 CYCLE
  INC ZPTEMP
  BNE TEST2         ;INCREMENT POINTER
  INC ZPTEMP+1

TEST2
  LDA END
  CMP ZPTEMP
  LDA END+1         ;CHECK TO SEE IF ENDED
  SBC ZPTEMP+1
  BCC DONE          ;BRANCH IF DONE
 ;
 ;CONTINUE NORMAL NMI ROUTINE
 ;
  JMP (VECTOR)
 ;
 ;
 ;FINISHED REMOVE WEDGE
 ;
 ;
DONE
  SEI
  LDA VECTOR
  STA NMIVEC        ;RESET NMI VECTOR TO
  LDA VECTOR+1      ;ORIGINAL VALUE
  STA NMIVEC+1
  CLI
  JMP (NMIVEC)
```

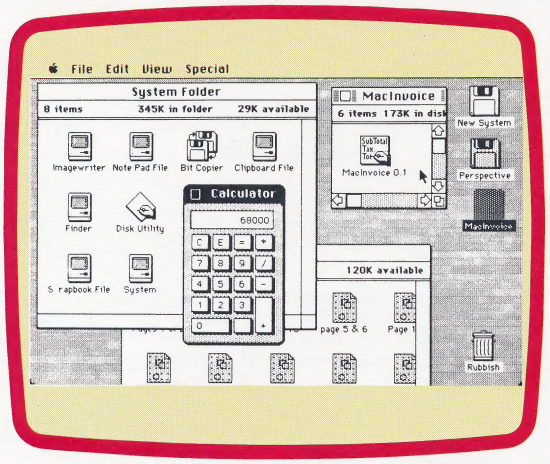
Parawedge, from 'Mastering the Commodore 64' by Jones and Carpenter, is reprinted by kind permission of the authors and Ellis Horwood Ltd.

OBJECT CODE

This is the output code from a compiler (see page 308). When the edited program file (known as the *source code*) is sent to the compiler, it will translate the high-level language in which the program has been written into a low-level language — for example, Assembly language. This low-level version of the program is the *object code* and it is this version that will be executed when the program is RUN. The advantage of an object program over one that needs interpreting is that the former, having already been translated into a low-level language, will run faster than, say, BASIC, which must be translated line by line into machine code before it is executed. Note, however, that compiled object code is not as efficient as a program written purely in machine code.

OBJECT-ORIENTED PROGRAMMING

Typically, a program consists of data and the operations, functions and procedures that are used to manipulate it. But in *object-oriented programming*, the data is regarded as objects, and the functions as messages. Whereas we normally specify within a program the steps required to manipulate the data, in object-oriented programming we merely specify, via the message, what we want to happen to the data. The computer then decides how best to carry out the message.



The Object At Hand
The Apple Macintosh was one of the first micros to put object-oriented programming into practice. Instead of having to issue a series of commands to the computer to perform an action, the user merely moves a cursor (via a hand-held mouse) to the appropriate icon, and presses a button on the mouse. The computer's operating system then carries out the command without the need for any further instructions

OCTAL NOTATION

This is a form of numeric notation set to base eight. Thus, in octal notation, numbers can be in the range of zero to seven. This system of notation was popular with older four-bit computers, since the whole range of numbers could be contained within three-bit binary patterns (111 in binary being equivalent to seven in octal), and octal 10 in four bits. However, with the advent of eight-bit microprocessor-based computers, octal notation has generally been superceded for most applications by hexadecimal (base 16) notation.

ONE'S COMPLEMENT

One's complement is an intermediary stage in producing the 'two's complement' (see page 328) method of binary arithmetic. In one's complement, we take an eight-bit binary number and substitute the ones for zeros and vice versa (in effect, performing a NOT operation on the number). Thus, binary 00010011 (decimal 19) becomes 11101100. By looking at this binary pattern using the convention of signed binary numbers, in which the leftmost bit is taken to refer to the 'sign' of this number, the fact that the leftmost bit is now 1 will indicate to the computer that the binary pattern is a negative number. The machine will then, using the one's complement system, regard the bit pattern as decimal -19.

OPERAND

Most commonly used when referring to machine code or Assembly language, an *operand* is one of two kinds of information (the other being operations such as adding and subtracting) held within the processor, on which logical or arithmetical operations contained in instructions may be performed.

OPERATING SYSTEM

The *operating system* is a vital component of the computer. Without it, all the processes and peripheral management would have to be performed manually. Essentially, an operating system is the software within the computer that provides an interface between you and the machine's electronics. There is a vast number of different operating systems on the market, but they fall into two main categories. There are the ROM-based systems that manufacturers build into their machines; these are what are usually found on most home computers.

There are also other types of operating systems, most commonly held on disk, which are loaded into the computer on power-up. Examples of this type include CP/M and MS-DOS. These operating systems are mostly, although not exclusively, disk operating systems. The advantage of these is that they are 'transportable', which means they can be transferred from one compatible machine to another. Because many different makes of machine use the same operating system, this, in theory, means that many software packages that run under the same operating system are equally portable.

OPERATION

An *operation* is a function that manipulates an input (or inputs) to produce an output (a result). The commonly used symbols in mathematics, such as +, − and ×, represent arithmetical operations, but apart from these types, there are several other operations that are widely used in computing. Binary operations, for example, are derived from the simple arithmetic function in which two numbers are added together. Another commonly used type is the logical operation, which will take up to two operands to produce a Boolean result. Examples of logical operations are NAND, OR and NOT.

THE INKING MAN'S MACHINE

As an exercise in understanding the fundamentals of computer programming, the use of relatively inexpensive robotic devices, or turtles, has gained wide popularity, especially in schools. We look at the Penman Plotter, which incorporates a wider range of functions than most comparable devices.

The large number of available interfaces on the BBC Micro, coupled with its wider adoption with educationalists, have combined to produce a wide number of educational peripherals for the machine. We have examined a large number of those devices that can be controlled by the BBC Micro, including such diverse machines as plotters, mice and floor robots. However, the Penman Plotter is a peripheral that can claim to be all three devices in one.

The Penman Plotter package consists of a control unit, a mobile plotter, a power supply, an RS232 connecting cable and accompanying software. When it is not being used, the control unit and plotter fit together to form a single compact unit, measuring only 55 by 128 by 335mm. To remove the plotter from its housing in the control unit, you need only to press the clip underneath it, slide the plotter out and unreel the connecting ribbon cable from inside the control unit itself. Once removed from its housing, the plotter reveals three sockets into which pens can be dropped, as well as a central well in the middle to accommodate another pen for turtle-type graphics. The 40mm–long felt tip pens, when dropped into their wells, are held above the paper by clips supported by springs. When it is commanded to draw, the spring will be pulled down by electromagnets attached to levers within, and the weight of the pen will force the top onto the paper.

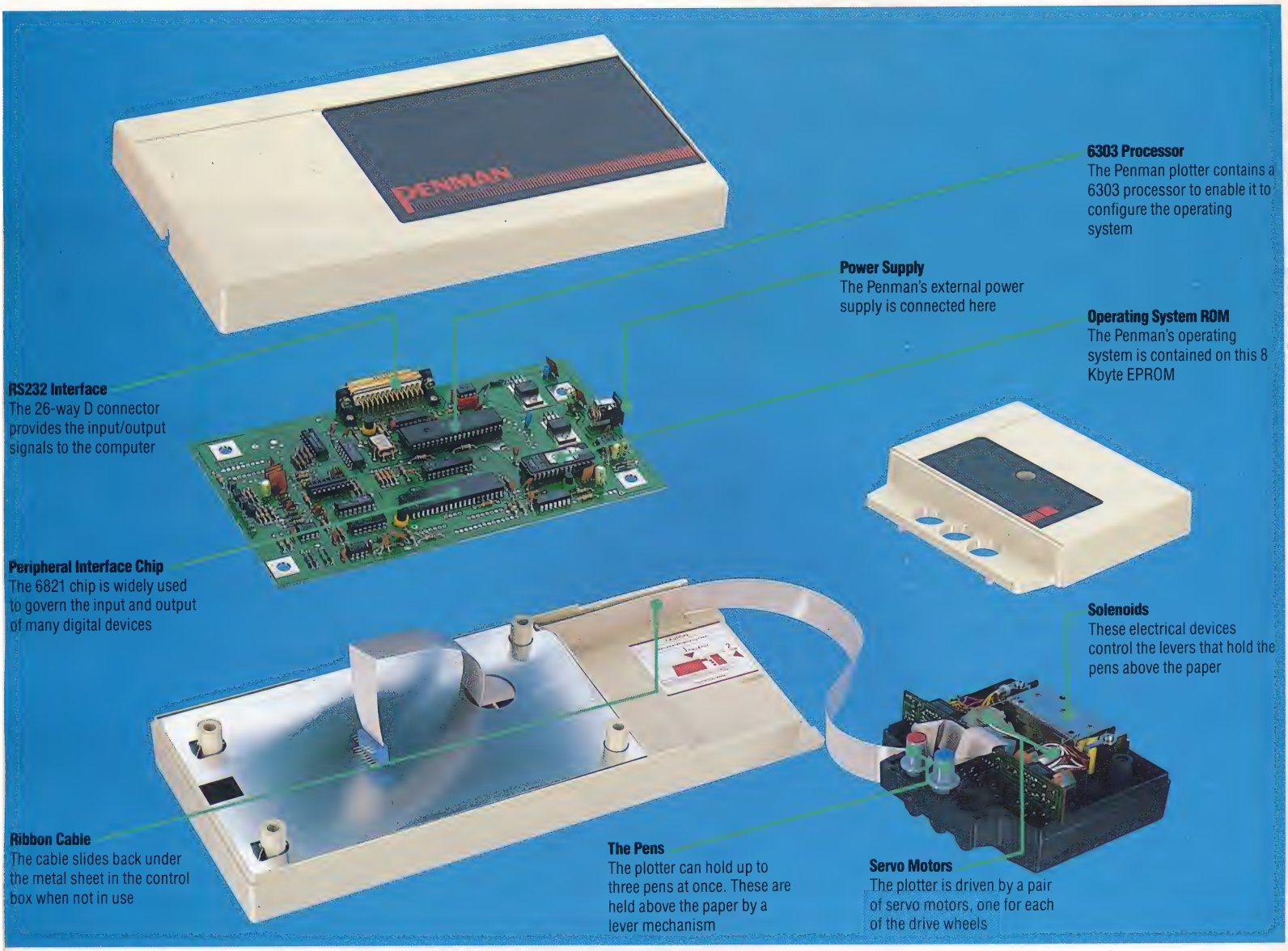
There are three wheels on the underside of the device, two of which are drive wheels; the third is a small plastic wheel on a free running caster to provide balance. The drive wheels are made of metal and are surrounded by a coarse casing to provide additional friction when the plotter is moving on paper. In front of each is a light sensor, which detects the edge of the paper by registering the difference in brightness between the paper and the material on which it is placed.

There are three electromagnets connected to levers inside the plotter, each of them controlling the raising and lowering of a pen onto the paper — the pair of standard electric motors attached to the drive wheels is adjacent. Ordinary electric motors are used by the plotter rather than servo or stepper motors; these allow the device to plot smooth curves rather than 'stepped' curves. However, this requires the software control to be correspondingly more sophisticated because it has to be able to vary the voltages applied to the motors, rather than having to send just a series of pulses.

Finally, there is a circuit board distributing the power and decoding the signals arriving from the control unit, and sending them to either the electromagnets or the motors. On each of the motor spindles, there is a stroboscopic disc that spins as the motor rotates, and on either side of the disc are light detecting diodes. As the stroboscopic disc spins, pulses of light appear on the diodes informing the logic within the circuit board on how

CHRIS STEVENS

Split Personality
The Penman Plotter consists of a control module, connected to a computer via an RS232 interface, and a moving turtle. The ribbon cable which joins the turtle to the control box is bi-directional, which means that the two devices can communicate with each other. Consequently, while the control box can send instructions to the turtle to move to a position, the turtle can transmit its position back to the control circuitry. It is this feature of the system that allows the Penman Plotter to be used not only as a turtle and plotter but also as a 'mouse'



fast the motor is moving so that the machine can work out its position. This logic is particularly useful when the Penman is being used as a 'mouse' because the computer needs to know in which direction and how fast the wheels are turning in order to move the cursor on the screen.

The printed circuit board inside the control unit has three main chips fitted. The board contains a 6303 microprocessor (a development of the 6800), which has the facility to configure a variety of operating systems. The processor incorporates a small 'scratchpad' RAM enabling it to store the plotter's position. The second major chip on the board is a 6821 peripheral interface ROM. Also included on the circuit board is an EPROM containing the demonstration programs the Penman will perform if the RS232C cable is not connected.

The Penman can be used in a variety of different modes. The 'terminal emulator' mode allows the device to be controlled directly from the keyboard. To enter it, you can either load the Pentlk program directly or, by using the Penman robot as a mouse, you can load the program from the main menu. Once the software is loaded, commands are sent to the control unit via the RS232 port in the form of ASCII codes. Typing PRINT 'I' will initialise the Penman. It will receive the signal and determine which of the three possible baud rates (300, 1200, 9600) is being used. Once this is done the Penman can be put through its homing sequence by typing the command H. The robot will go through a sequence of movements that result in the device locating the bottom left-hand corner of the paper.

From its starting position, the Penman will try to find the bottom of the page by using its light detecting sensors. Once the edge has been found, the device will make a 90° turn and perform the same action along the left of the page. As an aid in ensuring that the contrast between the paper and its background is sufficient to be detected by the plotter, Penman Products has thoughtfully included a piece of black paper onto which the drawing paper can be placed. After the Penman has 'homed', it sets its origin position 50mm from each edge of the paper (the distance from the front of the plotter to the central well).

The Pentlk application can be run in either

direct mode (one command at a time), or commands can be strung together to form programs that can be LOADed, SAVEd or RUN. Movement in 'terminal emulator' mode is Cartesian — meaning that the paper on which the robot is placed is divided into a grid. When the plotter is given an instruction to go to (500, 500), it will move to those co-ordinates rather than move 500 steps both ways. An A4 piece of paper has a maximum of 2,100 co-ordinates on the x axis and 2,970 on the y axis. Commands can be entered in either absolute or relative mode, depending on whether the MOVE instruction is prefixed either by an A or R. Pens are selected with LOGO-like commands such as U for pen up, D for pen down and P for pen select.

The Penman Plotter can also produce turtle graphics movements in direct mode, although these are somewhat more complex than Cartesian movements, in that the length of the distance to be moved must be entered in hexadecimal notation and prefixed by a $. This is because the turtle graphics bypasses the normal software that interprets Cartesian movements and directly interrogates the bits in the controlling addresses within the computer. However, this is not the only way that you can perform turtle graphics with the Penman Plotter. The applications software provided contains programs allowing the robot to be controlled from LOGO.

CONTROL IN ROBOTICS MODE

A similar system of directly controlling the Penman from the user port's data direction register is used in controlling the plotter in robotics mode. Each bit of the register controls a different aspect of the robot's movements — bits 0 and 1 and bits 2 and 3 control the right and left motors respectively. Bits 4 and 5 perform general motor functions, such as switching the motors on and off, while bit 6 controls the up and down pen movement. The Penman also returns information about itself to the data register, such as positional errors and whether or not the plotter has detected the edge of the page with its light sensors.

CHRIS STEVENS

Text commands can also be sent to the Penman Plotter. The range of commands and their applications are very similar to those used on the printer/plotters available on a wide number of home machines (see page 731). In fact, the shape of the printed characters from each bear remarkable similarity. Text can be varied in size from one to 127mm in height and can be printed in any one of four directions. A useful addition on the Penman Plotter that is not available on conventional printer/plotters is its ability to slant the text.

The manual accompanying the plotter provides a full list of the available commands, with some explanation of how they are implemented, as well as an explanation of the hardware. It is perhaps a little advanced for the beginner, however, and it seems the Penman Plotter is aimed at those who already possess a thorough knowledge of the workings of their computer.

The Penman Plotter can be considered to be another advance in the development of an all purpose turtle/plotter. When used correctly, the resolution of the device begins to approach that of more conventional plotters used in commercial design studios. However, at the moment, the software prevents the device from being a viable business tool. The current suite of programs provided with the plotter is intended primarily as an educational tool. Users are expected to write their own programs to run the plotter, thus learning the principles of robotics software. In contrast, a commercial user will not particularly care about the finer details of programming, but will require something that is easy to use.

As an educational device, the Penman Plotter appears good value. Although not the cheapest device of its type on the market, the range of operation modes available makes the machine an attractive proposition to schools, where it can be used to demonstrate a variety of applications.

PENMAN PLOTTER

PRICE
£249

DIMENSIONS
335×128×55mm

INTERFACES
RS232C, allowing it to be connected to any RS232-compatible computer

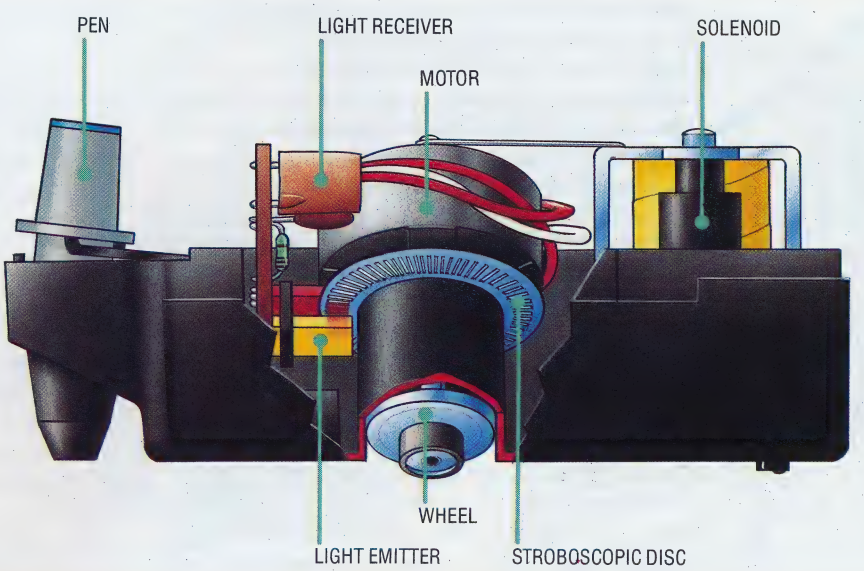
RESOLUTION
Text size: 1 to 127mm; graphics resolution: 0.1mm

DOCUMENTATION
The manual provides a lot of technical detail that will enable advanced users to develop the potential of the Penman, although it lacks enough tutorial material for the beginner

As The Wheel Turns

To enable the Penman control system to 'know' the position of the plotter, there must be some sort of feedback mechanism within the robot to count the number of 'steps' that have been moved. To do this the manufacturers have attached a thin metal disc around each of the wheel drive shafts. The disc has a number of thin slits cut from the centre towards the edge.

Mounted on the circuit board is a device that emits and receives light. As the disc spins, a stroboscopic effect is produced between the emitter and receiver as a series of pulses. The number of pulses will be calculated to determine the number of steps taken



KEVIN JONES

ALONE ON A WIDE WIDE SEA

As our ship sails on its course for the New World, several minor 'events' will affect the duration of the voyage and the strength of the crew. We conclude the programming of these events in this instalment with a module dealing with catching fish and collecting fresh water, as well as experiencing bouts of fine sailing weather.

The code for the remaining minor contingencies is dealt with in a series of small program sections that are selected at random by the subroutine at line 5500. In the previous instalment, we dealt with the first five events, so when we add the new events, there will be a total of 13, necessitating the changing of the value of RM, in line 46, to 13. The list of line numbers, following the ON X GOTO statement in line 5525, must also be increased to accommodate the new routines. The main program selects one minor incident randomly from its extended list to execute each week.

```
5525 ON X GOTO 5540,5570,5570,5570,5570,5600,5700,
5800,5850,5900,5950,6000,6050
```

The first line number to be added to the ON . . . GOTO statement in line 5525 is 5600 — the routine concerned with catching fish begins here. If X is determined to be 6 by the random event generator, then line 5525 will send the program to this routine. When a crew member dies, the death is not recorded in variable CN until the end of the week; but if the remaining crew have passed away during the current week, then there will, of course, be no one available to catch any fish.

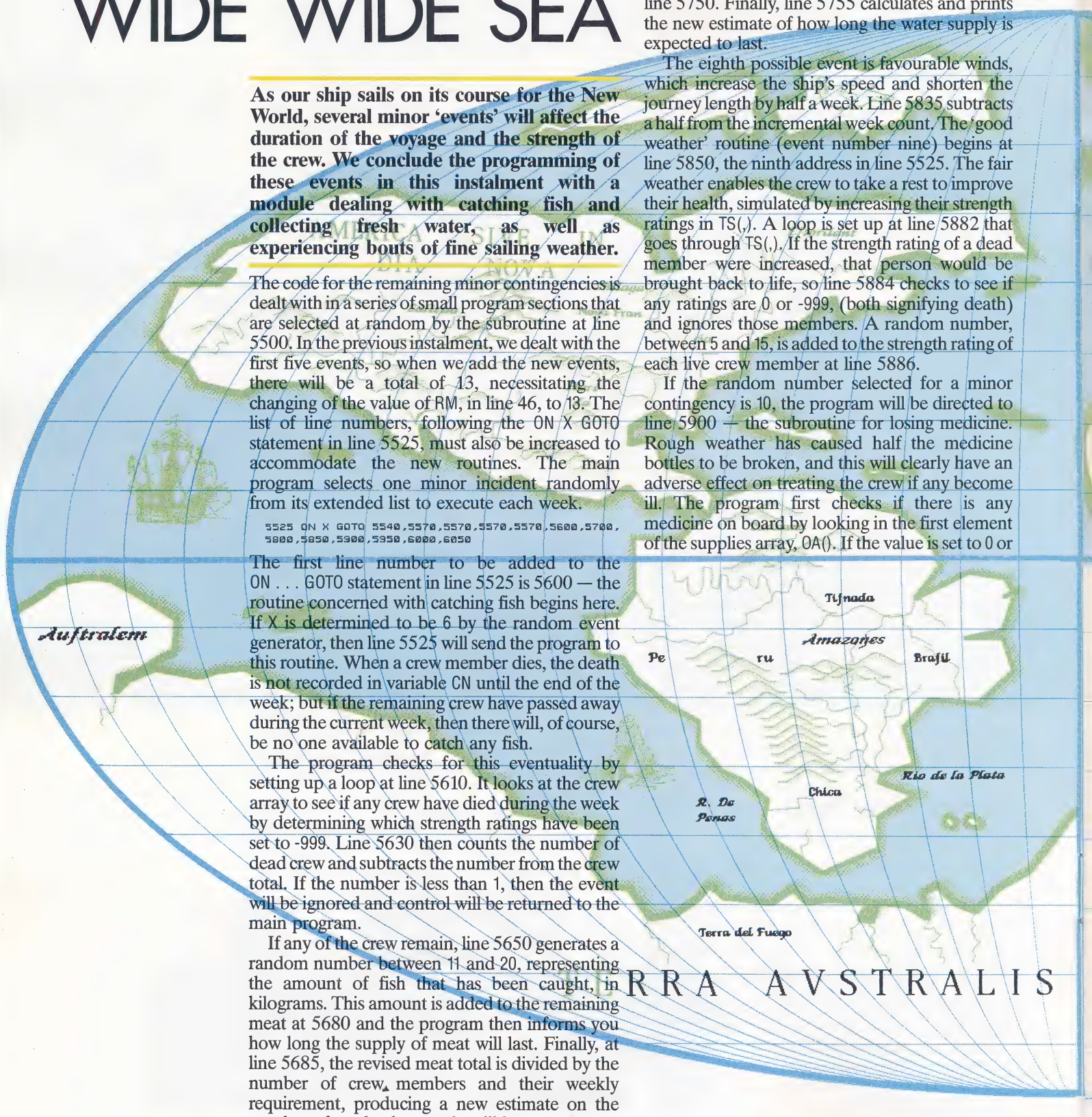
The program checks for this eventuality by setting up a loop at line 5610. It looks at the crew array to see if any crew have died during the week by determining which strength ratings have been set to -999. Line 5630 then counts the number of dead crew and subtracts the number from the crew total. If the number is less than 1, then the event will be ignored and control will be returned to the main program.

If any of the crew remain, line 5650 generates a random number between 11 and 20, representing the amount of fish that has been caught, in kilograms. This amount is added to the remaining meat at 5680 and the program then informs you how long the supply of meat will last. Finally, at line 5685, the revised meat total is divided by the number of crew members and their weekly requirement, producing a new estimate on the number of weeks the supply will last.

The next event in the ON X GOTO statement is a rainstorm. Starting at line 5700, the crew manage to catch some rain in barrels and replenish their water supply. The program generates a random number between 11 and 20 at line 5735 representing the number of barrels of collected water, which is added to the existing number by line 5750. Finally, line 5755 calculates and prints the new estimate of how long the water supply is expected to last.

The eighth possible event is favourable winds, which increase the ship's speed and shorten the journey length by half a week. Line 5835 subtracts a half from the incremental week count. The 'good weather' routine (event number nine) begins at line 5850, the ninth address in line 5525. The fair weather enables the crew to take a rest to improve their health, simulated by increasing their strength ratings in TS(,). A loop is set up at line 5882 that goes through TS(,). If the strength rating of a dead member were increased, that person would be brought back to life, so line 5884 checks to see if any ratings are 0 or -999, (both signifying death) and ignores those members. A random number, between 5 and 15, is added to the strength rating of each live crew member at line 5886.

If the random number selected for a minor contingency is 10, the program will be directed to line 5900 — the subroutine for losing medicine. Rough weather has caused half the medicine bottles to be broken, and this will clearly have an adverse effect on treating the crew if any become ill. The program first checks if there is any medicine on board by looking in the first element of the supplies array, OA(). If the value is set to 0 or



- 999, there is no medicine, and so this contingency is ignored. If there is medicine aboard, line 5925 halves the amount; since bottles do not break by halves, however, it takes the integer result. The program then prints a message informing the player how many bottles of medicine remain.

The eleventh contingency occurs when a freak wave causes some of the guns to get wet and rust, rendering them useless. An important commodity, guns may be needed for defence, acquiring food and trading. The subroutine for rusty guns is similar to that for broken medicine bottles. It checks if there are any guns on board, at line 5955, halves the amount, at line 5975, and prints the integer result.

Having half the bales of cloth eaten by mice represents the twelfth event, starting at line 6000. Cloth is the fourth element in the supply array OA(), so the program checks OA(4) to see if there is any cloth aboard at line 6005. If cloth has been purchased, the mice, who reside at line 6025, eat half of it, after which feast, the player is informed of the amount of bales that remain.

Note that there is an opportunity at this stage to tinker with the game, making it easier or more difficult by altering factors in the contingencies. The limits between which random numbers are selected can be reduced or extended and the proportions of supplies lost can be similarly increased or decreased.

ALBATROSS SIGHTING

The final minor contingency, ominously numbered 13, is the sighting of an albatross. This event is unique amongst the minor contingencies, as there are two possibilities of it happening during one week. If the ON X GOTO statement selects the thirteenth event, the incident will occur in the normal way, as a minor contingency. If the albatross is sighted when the crew are on half rations of meat, they will be given the opportunity to shoot it for food. Players who are familiar with Coleridge's poem, *The Rime of the Ancient Mariner,* will be justly wary of doing this. In the main program, the minor contingencies are called by the GOSUB at line 860; the major contingencies will start at line 870. At 875, the second possibility of seeing the albatross occurs. This line checks if the crew are on half rations of meat, by checking if HR(3) equals 0.5. If so, a random factor — AND RND(1) <.5 — gives a 50 per cent chance of a sighting. If the bird is sighted under these circumstances, the program goes to the 'albatross' subroutine, which starts at line 6050.

Sighting an albatross brings good luck. A variable, A$, is set to 'Y' at line 6055, indicating that the bird has been spotted. This will be used later in the program to reduce the possibility of a mutiny, a practical result of good luck. The program then checks, at line 6075, if the crew are running short of meat, by determining if the amount of meat remaining is equal to, or greater than, the weekly requirements of the crew multiplied by the remaining journey time. If supplies are adequate, the bird flies away and line 6130 returns to the main program.

If meat stocks are low, the player is told that the bird weighs 10 kilos, and asked if he wants to shoot it. Line 6115 checks if this is so, by looking at the first character of the input. If the player decides not to fire, the bird flies away; but if he does want to shoot the bird, the program checks — at line 6133 — if OA(2) is set to 0 (indicating that there are no guns). If it is set to -999, the guns have been lost during the current week and the program informs the player, and the bird flies away.

If there are guns on board, a shot is fired. Line 6140 gives the player a 50 per cent chance of hitting the bird, by generating a random number between 0 and 1. If the number is greater than .5 the bird is missed and it flies away; otherwise the bird is hit and falls to the deck. If the supplies of meat were exhausted during the current week, the corresponding array value is reset to 0, from -999, at line 6160, to enable the extra provision provided by the bird's carcass to be recorded. The amount of meat in the provision array, PA(3), is increased by 10 kilos and the player is told the new total.

The shooting of the albatross in Coleridge's poem is signalled by an interruption from the wedding guest:

'God save thee, ancient Mariner!
From the fiends that plague thee thus.
Why lookst thou so?'
'With my cross-bow
I shot the albatross.'

A similar warning is given to the player in our

IAN McKINNELL ON THE MACINTOSH

game. In line 6162, B$ is set to 'Y'. This is the bad luck factor, which more than negates the good luck brought in line 6055. The precise nature of this luck will be revealed later in the course. The player will not be informed of the reason, but factors will be changed to hinder the ship's progress.

Module Seven: More Random Events

Addition To Main Voyage Loop

```
875 IF HR(3)=.5ANDRND(1)<.5THENPRINTCHR$
(147):GOSUB6050
```

More Random Events Subroutine

```
5600 REM EVENT 6 - CATCH SOME FISH
5605 X=0
5610 FORT=1TO16
5615 IFTS(T,2)=-999THENX=X+1
5620 REM COUNT DEAD THIS WEEK
5625 NEXT
5630 IFCN-X<1THENRETURN
5635 REM NO ACTION IF ALL CREW DEAD
5640 PRINT
5645 S$="  DURING THE WEEK*":GOSUB9100
5646 PRINT:GOSUB9200
5650 S$="ONE OF THE CREW CAUGHT*":GOSUB9100
5655 X=INT(RND(1)*10)+11
5660 REM 10 TO 20 KILOS
5662 PRINTX;"KILOS OF FISH"
5665 PRINT:GOSUB9200
5670 S$="YOUR MEAT SUPPLY IS NOW*":GOSUB9100
5675 S$="ENOUGH FOR APPROXIMATELY*":GOSUB9100
5678 IFPA(3)=-999THENPA(3)=0
5680 PA(3)=PA(3)+X
5685 PRINTINT(PA(3)/(CN*PN(3)));"WEEKS"
5690 GOTO5530
5700 REM EVENT 7 - CATCH SOME WATER
5705 PRINT
5710 S$="  DURING THE WEEK*":GOSUB9100
5715 PRINT:GOSUB9200
5720 S$="A RAINSTORM REFILLED YOUR*":GOSUB9100
5725 S$="WATER BARRELS*":GOSUB9100
5730 PRINT:GOSUB9200
5735 X=INT(RND(1)*10)+11
5736 REM 10 TO 20 BARRELS
5740 S$="YOUR WATER SUPPLY IS NOW*":GOSUB9100
5745 S$="ENOUGH FOR APPROXIMATELY*":GOSUB9100
5748 IFPA(4)=-999THENPA(4)=0
5750 PA(4)=PA(4)+X
5755 PRINTINT(PA(4)/(CN*PN(4)));"WEEKS"
5760 GOTO5530
5800 REM EVENT 8 - GOOD WINDS
5805 PRINT
5810 S$="STRONG FOLLOWING WINDS ALL WEEK*":GOSUB91
00
5815 PRINT:GOSUB9200
5820 S$="YOU HAVE MADE GOOD SPEED*":GOSUB9100
5825 S$="AND YOUR JOURNEY TIME IS*":GOSUB9100
5830 S$="REDUCED BY 1/2 A WEEK*":GOSUB9100
5835 EW=EW-.5
5839 GOTO5530
5850 REM EVENT 9 - GOOD WEATHER
5855 PRINT
5860 S$="GOOD WEATHER ALL WEEK*":GOSUB9100
5865 PRINT:GOSUB3200
5870 S$="THE CREW IS FEELING HAPPIER*":GOSUB9100
5875 GOSUB9200
5880 S$="                AND HEALTHIER!*":GOSUB9100
5882 FORT=1TO16
5884 IFTS(T,2)=0 OR TS(T,2)=-999THEN5888
5886 TS(T,2)=TS(T,2)+INT(RND(1)*11)+5
5888 NEXT
5889 GOTO5530
5900 REM EVENT 10 - LOSE MEDICINE
5905 IF OA(1)=0 OR OA(1)=-999THENRETURN
5910 PRINT
5915 S$="YOU DISCOVER THAT HALF YOUR*":GOSUB9100
5920 S$="MEDICINE BOTTLES HAVE BROKEN*":GOSUB9100
5925 OA(1)=INT(OA(1)/2)
5930 PRINT:GOSUB9200
5935 S$="YOU NOW HAVE ONLY*":GOSUB9100
5940 PRINTOA(1);"BOTTLES LEFT"
5945 GOTO5530
5950 REM EVENT 11 - RUSTY GUNS
5955 IF OA(2)=0 OR OA(2)=-999THENRETURN
5960 PRINT
5965 S$="YOU DISCOVER THAT HALF YOUR*":GOSUB9100
5970 S$="GUNS HAVE GONE RUSTY*":GOSUB9100
5972 S$="AND NO LONGER WORK*":GOSUB9100
5975 OA(2)=INT(OA(2)/2)
5980 PRINT:GOSUB9200
5985 S$="YOU NOW HAVE ONLY*":GOSUB9100
5990 PRINTOA(2);"GUNS LEFT"
5995 GOTO5530
6000 REM EVENT 12 - LOSE CLOTH
6005 IF OA(4)=0 OR OA(4)=-999THENRETURN
6010 PRINT
6015 S$="YOU DISCOVER THAT HALF YOUR*":GOSUB9100
6020 S$="BALES OF CLOTH HAVE BEEN*":GOSUB9100
6022 S$="GNAWED BY MICE*":GOSUB9100
6024 S$="AND ARE NOW WORTHLESS*":GOSUB9100
6025 OA(4)=INT(OA(4)/2)
6030 PRINT:GOSUB9200
6035 S$="YOU NOW HAVE ONLY*":GOSUB9100
6040 PRINTOA(4);"BALES LEFT"
6045 GOTO5530
6050 REM EVENT 13 - ALBATROSS
6055 PRINT:A$="Y"
6060 S$="AN ALBATROSS FLIES OVER THE SHIP*":GOSUB9
100
6062 GOSUB9200
6065 S$="THIS IS A GOOD OMEN*":GOSUB9100
6068 S$="AND THE CREW IS HAPPY*":GOSUB9100
6070 PRINT:GOSUB9200
6075 IFPA(3)<(CN*PN(3)*(JL-WK+1))THEN6090
6080 REM NOT SHORT OF MEAT
6085 GOTO6122
6090 S$="YOU'RE RUNNING SHORT OF MEAT*":GOSUB9100
6095 S$="AND THE BIRD WEIGHS 10 KILOS!*":GOSUB9100
6100 PRINT:GOSUB9200
6105 S$="WOULD YOU LIKE TO CATCH IT?*":GOSUB9100
6110 INPUTI$
6112 PRINT:GOSUB9200
6115 IFLEFT$(I$,1)="Y"THEN6133
6120 S$="PROBABLY JUST AS WELL!*":GOSUB9100
6122 PPINT:GOSUB9200
6125 S$="THE ALBATROSS FLIES AWAY....*":GOSUB9100
6130 GOTO5530
6133 IFOA(2)=0OROA(2)=-999THEN6180
6135 S$="A SHOT IS FIRED.............*":GOSUB9100
6138 GOSUB9200:GOSUB9200
6140 IFRND(1)<.5THEN6150
6145 S$=".........BUT MISSES!*":GOSUB9100
6148 GOTO6122
6150 S$="AND THE BIRD FALLS TO THE DECK!*":GOSUB91
00
6155 PRINT:GOSUB9200
6160 IFPA(3)=-999THENPA(3)=0
6162 PA(3)=PA(3)+10:B$="Y"
6165 S$="YOU NOW HAVE 10 MORE KILOS*":GOSUB9100
6167 S$="OF MEAT......*":GOSUB9100
6170 S$="BUT YOU MAY NOT HAVE MUCH*":GOSUB9100
6172 S$="GOOD LUCK FROM NOW ON!!*":GOSUB9100
6174 GOTO5530
6180 S$="YOU CAN'T - YOU HAVE NO GUNS*":GOSUB9100
6190 GOTO6122
```

Basic Flavours

Spectrum:

Make the following changes:

```
875  IF HR(3)=.5ANDRND(1)<.5THENCLS
     :GO SUB 6050
5527 IF X=6 THEN GO TO 5600
5528 IF X=7 THEN GO TO 5700
5529 IF X=8 THEN GO TO 5800
5530 IF X=9 THEN GO TO 5850
5531 IF X=10 THEN GO TO 5900
5532 IF X=11 THEN GO TO 5950
5533 IF X=12 THEN GO TO 6000
5534 IF X=13 THEN GO TO 6050
5535 PRINT:S$=K$:GO SUB 9100
5536 LET I$=INKEY$:IF INKEY$="" THEN GO
     TO 5536
6115 IF I$(1 TO 1)="Y" THEN GO TO 6133
```

BBC Micro:

Make the following changes:

```
875  IF HR(3)=.5AND RND(1)<.5
     THEN CLS: GOSUB6050
```

AT ARM'S LENGTH

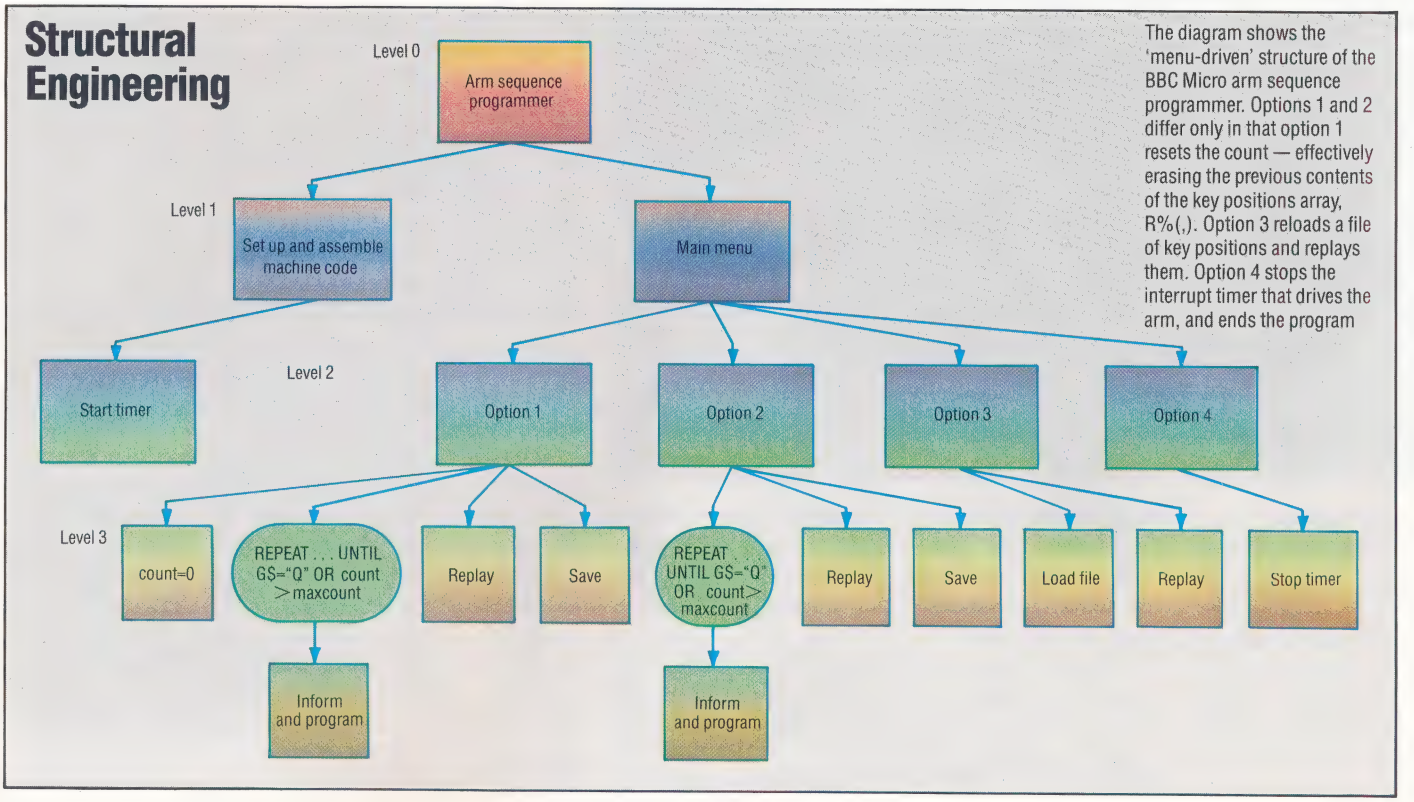
In the last instalment of Workshop, we completed the assembly of the robot arm. We can now design the software that will control the arm movements using the BBC Micro, Commodore 64 and the Sinclair Spectrum. In this instalment, we focus our attention on the BBC Micro.

The four motors that move the arm are connected to and controlled by four data lines from the computer with which the arm is interfaced. In the case of the BBC Micro these four data lines emanate from the user port using pins D0 to D3. Because servos rely on a continuous stream of pulses to tell them what angle to take up, they cannot be controlled from BASIC alone but must be driven by a machine code wedge program, which sends a new set of signals to the motors every one sixtieth of a second using an interrupt. (The principles of this control method are given in more detail on pages 912 and 923.) The multiple servo control program given on page 925 can be used to test the operation of the arm directly from the keyboard, and though not very sophisticated, will test each motor independently.

Once the wedge code is running in the background, it will then be possible to break out of the BASIC key-scanning program to place numbers directly into the locations that control the motor angles. The address of the first motor-angle memory location is angle; motor 0 is set using the value in this location. Location angle + 1 sets the position of motor 1, and so on. Remember that motor 0 controls movement from the waist, motor 1 from the shoulder, motor 2 movement from the elbow, and motor 3 opens and closes the grab mechanism. If any angle is incremented or decremented by one unit, then the motors can easily follow. If large changes, such as 0 to 128, are introduced, then the motors may take a moment or two to react. Also, notice how the combined effects of inertia and sponginess in the arm make it stop with a judder, reminiscent of the robotic dance craze.

PROGRAMMING MOVEMENT SEQUENCES

Having tried out the arm using the simple control program, it becomes apparent that a more sophisticated program is needed. Deciding exactly how to program the arm, however, is probably the most difficult part of the whole operation. Many industrial robots are programmed by moving the grab manually through the sequence that it has to learn; the arm will automatically record key

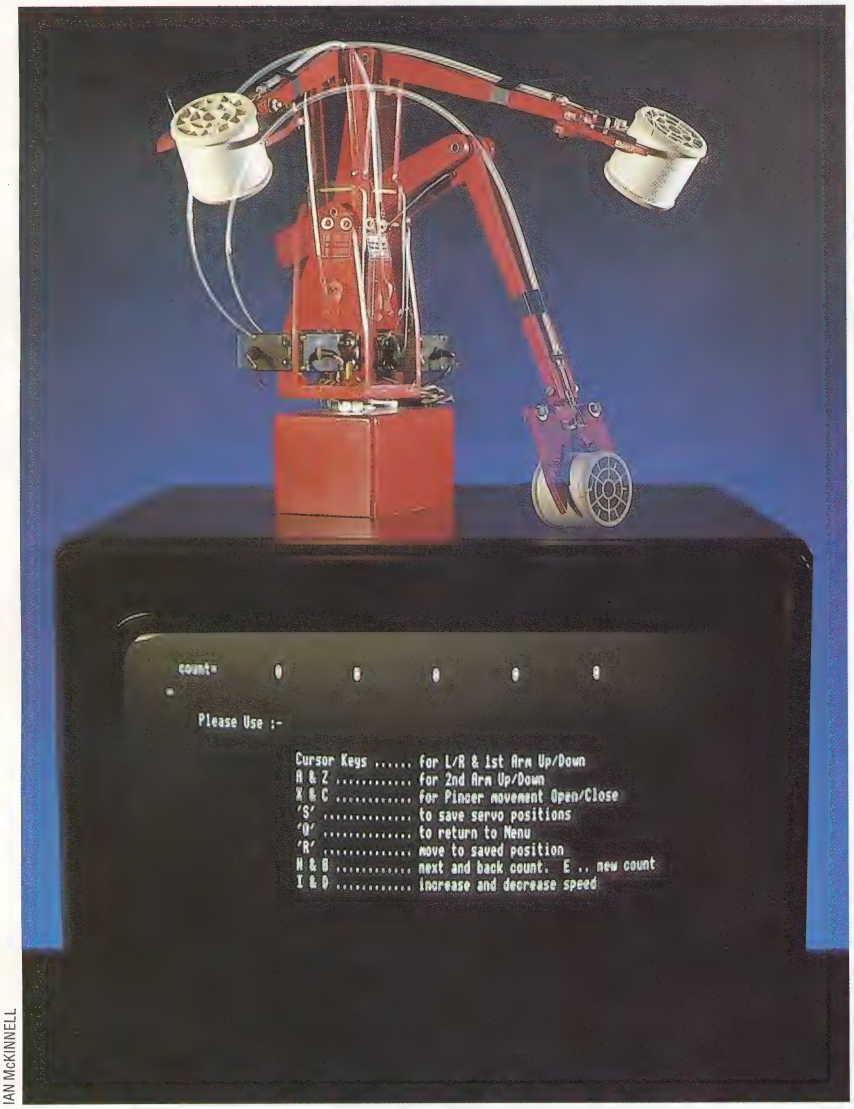


The diagram shows the 'menu-driven' structure of the BBC Micro arm sequence programmer. Options 1 and 2 differ only in that option 1 resets the count — effectively erasing the previous contents of the key positions array, R%(,). Option 3 reloads a file of key positions and replays them. Option 4 stops the interrupt timer that drives the arm, and ends the program

KEVIN JONES

positions in the sequence. Unfortunately, this method requires rather sophisticated pieces of hardware such as angular feedback sensors, which our robot does not possess.

But even with sophisticated equipment, such as that used in industrial applications, the arm may be working in hazardous environments in which an operator cannot physically go to program the arm. Therefore, an alternative method is to control the arm remotely from the keyboard, sending it through a series of movements and saving key positions in the sequence by pressing another key, and recording the four motor angles that define each position in an array. The sequence may be replayed simply by stepping through the array of key positions with a suitable delay factor. Sequences can also be saved to or loaded from disk or tape by using a sequential file to hold the contents of the array. This is the method of control that we will use.



IAN McKINNELL

Picking Up The Pieces
We started the robot arm project by considering a particular task that we wanted to perform: to pick up and move an object the size and weight of a cotton reel. The arm sequence programmer allows us to program the arm to do this task, if necessary saving the movements onto a file that can be reloaded and replayed at any time

The first stage in the design of our software is, therefore, to assign keys to control the various arm functions so that a sequence of movements can be 'taught' to the arm by moving it remotely from the keyboard. The BBC Micro's cursor keys can be re-assigned to control the left/right movement of the main body and the up/down movement of the lower arm. The A and Z keys will be used to move the upper arm up and down and the X and C keys to open and close the grab jaws. The I and D keys allow the speed of the arm to be increased or decreased, so that delicate manoeuvres can be performed slowly, while larger, sweeping movements are carried out quickly. The S key allows the current arm position to be stored in the key position array, and R returns the arm to the last stored position. The sequence can be stepped through to make changes — either forwards or backwards using the N and B keys, respectively; E controls movements to certain points within the sequence. By using these last three keys, any programmed sequence of positions can be edited. The procedures inform and program in the listing provide you with a list of the key functions as well as repeatedly scanning the keyboard for input.

The other procedures, replay and save, allow the sequence held in the key positions array, R%(,), to be replayed or saved to disk or tape, whilst loadfile re-assigns the values in R%(,) using values previously saved to a sequential file. Thus, taken as a whole, the program gives us a comprehensive method of programming, editing and replaying arm movement sequences.

A SMOOTHER MOVEMENT

The machine code interrupt-handling routine controls the servo motor angles by looking at four locations, starting at address angle. Each of these locations contains a value corresponding to the angle that each motor should take up. One of the reasons the original test program (see page 925) produces jerky movements is that the BASIC program POKEs values directly into these locations. Large changes in an angle produce a violent movement by the motor as it tries to attain the new position as quickly as possible. To make the arm move more smoothly, we should try to adjust the values in the four angle locations by small amounts only. For this reason, a second group of locations, starting at newpos%, is used to accept the angle changes from the BASIC program. A short machine code routine can then be added to the program to increase or decrease the values in the four angle locations (in steps of one unit) until they match the values in the corresponding newpos% location. The procedure moveservo simply calls the machine code program that does this.

Introducing this new routine has an extra benefit. By inserting a delay loop within this piece of machine code, we can slow down or speed up the rate at which the motor moves from its old position to its new position, simply by altering the value that terminates the delay loop. Location &81 holds this delay factor that can be altered from BASIC at the beginning of the procedure replay, allowing previously programmed movement sequences to be implemented at different speeds. The machine code for this new routine starts at line 2000 in the given listing.

BBC Micro Arm Sequence Programmer

```
   10 REM **********************
   11 REM **********************
   12 REM **                    **
   13 REM ** BBC ARM SEQUENCE **
   14 REM **    PROGRAMMER     **
   15 REM **                    **
   16 REM **********************
   17 REM **********************
   20 :
   25 MODE 3
   30 PROCsetup
   40 CLS:PRINT TAB(5,6)"Would you like to : "
   50 PRINT TAB(25,10)"1.........Program new arm sequence"
   60 PRINT TAB(25,12)"2.........Add moves to program"
   70 PRINT TAB(25,14)"3.........Replay a file"
   80 PRINT TAB(25,16)"4.........Leave Program"
   90 G$=GET$:IF G$="1" THEN count=0
  100 IF G$="1" OR G$="2" THEN PROCinform:PROCprogram:PROCreplay:PROCsave
  110 IF G$="3" THEN PROCloadfile:PROCreplay
  120 IF G$="4" THEN CLS:PROCendtimer:END
  130 GOTO 40
  140
  150
  160 DEF PROCinform:CLS
  170 PRINTTAB(5,5)"Please Use :- "
  180 PRINTTAB(20,7)"Cursor Keys ..... for L/R & 1st Arm Up/Down"
  190 PRINTTAB(20)"A & Z ............ for 2nd Arm Up/Down"
  200 PRINTTAB(20)"X & C ............ for Pincer movement Open/Close"
  210 PRINTTAB(20)"'S' .............. to save servo positions"
  220 PRINTTAB(20)"'Q' .............. to return to Menu"
  230 PRINTTAB(20)"'R' .............. move to saved position"
  240 PRINTTAB(20)"N & B ............ next and back count.  E .. new count"
  250 PRINTTAB(20)"I & D ............ increase and decrease speed"
  260  PRINTTAB(2,2)"count="count"  ";:FOR I%=0TO3:PRINTR%(count,I%);"  ";:NEXT:
PRINT
  270 ENDPROC
  280
  290
  300 DEF PROCprogram:?&81=1
  310 REM re-assign cursor move keys
  320 *FX4,1
  330 DX=2
  340 REPEAT  :REM scan keys
  350  IF INKEY(-38)THEN DX=DX+1 :REM "I"
  360  IF INKEY(-51)THEN DX=DX-1 :IF DX<1 THEN DX=1 :REM "D"
  380  IF INKEY(-26)THEN newpos%?0=(newpos%?0)+DX: REM "LEFT ARROW" servo 0  rig
ht
  390  IF INKEY(-122)THEN newpos%?0=(newpos%?0)-DX:REM "RIGHT ARROW" servo 0 lef
t
  400  IF INKEY(-58)THEN newpos%?1=(newpos%?1)+DX:REM "UP ARROW" servo 1 up
  410  IF INKEY(-42)THEN newpos%?1=(newpos%?1)-DX:REM "DOWN ARROW" servo 1 down
  420  IF INKEY(-66)THEN newpos%?2=(newpos%?2)-DX:REM "A" servo 2 up
  430  IF INKEY(-98)THEN newpos%?2=(newpos%?2)+DX:REM "Z" servo 2 down
  440  IF INKEY(-67)THEN newpos%?3=(newpos%?3)+DX*3:REM "X" servo 3 open jaw
  450  IF INKEY(-83)THEN newpos%?3=(newpos%?3)-DX*3:REM "C" servo 3 close jaw
  460  IF INKEY(-52) THEN FOR I%=0TO3:newpos%?I%=R%(count,I%):NEXT :REM "R" repl
ay position
  470  IF INKEY(-86)THEN count=count+1:IF count>maxcount THEN count=0
  480  IF INKEY(-101)THEN count=count-1:IF count<0 THEN count=maxcount
  490  IF INKEY(-35) THEN PRINTTAB(10,20)"GIVE VALUE FOR count ";:INPUT count :R
EM "E" keyin new count
  500  IF INKEY(-82)THEN FOR I%=0 TO Noservoes%:R%(count,I%)=newpos%?I%:NEXT:cou
nt=count+1 :REM "S" save position at current count
  510 IF count>lastmaxc THEN lastmaxc=count
  520  IF oldc<>count THEN PRINTTAB(2,2)"count="count"  ";:FOR I%=0TO3:PRINTR%(c
ount,I%);"  ";:NEXT:PRINT
  530  oldc=count
  540 FOR I%=0 TO 3:IF newpos%?I%>200 THEN newpos%?I%=200
  550 IF newpos%?I%<10THEN newpos%?I%=10
  560 NEXT
  570 PROCmoveservo
  580 G$=INKEY$(1):UNTIL G$="Q" OR count>maxcount
  590 *FX4,0
  600 ENDPROC
  610
  620
  630 DEF PROCreplay
  640 REPEAT
  650 PRINTTAB(11,23)"Would you like a replay of movements ? (Y/N) R to repeat"
  660 G$=GET$
  670 UNTIL G$="Y" OR G$="N" OR G$="R"
  680 IF G$="N" THEN ENDPROC
  690 IF G$<>"R" THEN INPUTTAB(11,22)"What delay period between response  (1 to
255) then (return)";dly%:IF dly%>255 OR dly%<1 GOTO 690
  700 ?&81=dly%
  710 FOR PosNo=0 TO lastmaxc:PRINTTAB(19,2)"No. of sequential position    =
";PosNo;"  ":FOR servo%=0 TO Noservoes%:newpos%?servo%=R%(PosNo,servo%):NEXT:PRO
Cmoveservo:NEXT
  720 GOTO640
  730
  740
  750 DEF PROCmoveservo
  760 CALLmoveservo  :REM the machine code version
  820 ENDPROC
  830
  831
  840 DEF PROCsave:CLS
  850 PRINT TAB(15,10)"Would you like to save the sequence on file ?  Y/N ":IF G
ET$<>"Y" THEN ENDPROC
  860 INPUT TAB(15,12)"Please name the File to save : ";file$ '':file$="D."+file
$
  870 PROCendtimer
  880 X=OPENOUT file$:PRINT#X,lastmaxc:FOR A=0 TO lastmaxc:FOR B=0 TO Noservoes%
:PRINT#X ,R%(A,B):NEXT:NEXT
  890 CLOSE#X
  900 PROCstarttimer
  910 ENDPROC
  920
  930
  940 DEF PROCloadfile:CLS
  950 INPUT TAB(15,10)"Please name the File to load : ";file$:file$="D."+file$
  960 PROCendtimer
  970 X=OPENIN file$ :INPUT#X,count:lastmaxc=count:FOR A=0 TO count:FOR B=0 TO N
oservoes%:INPUT#X ,R%(A,B):NEXT:NEXT
  980 oldc=0
  990 CLOSE#X
 1000 PROCstarttimer
 1010 ENDPROC
 1020
 1030
 1040 DEF PROCsetup
 1050 count=0:Noservoes%=3:REM NO OF MOTORS-1
 1060 lastmaxc=0 :oldc=0 :dly%=10
 1070 maxcount=100
 1080 DIM R%(maxcount,Noservoes%),newpos% 8 ,timer% 12 ,read% 12
 1090 PROCassembletime
 1100 ENDPROC
 1110
 1120
 1130 DEF PROCassembletime
 1140 REM ****************************************
 1150 REM      Set up the timer etc
 1160 REM ****************************************
 1170 osbyte=&FFF4
 1180 A%=&97 :X%=&62 :Y%=&FF
 1190 CALL osbyte:REM set up port B for output
 1200 DIM p%(8)
 1210 DIM timer% 12 ,read% 12
 1220 xtimer=timer% MOD 256
 1230 ytimer=timer% DIV 256
 1240 xread=read% MOD 256
 1250 yread=read% DIV 256
 1260 PROCinitial
 1270 FOR I%=angle TO angle+8:angle?I%=128:NEXT
 1280 t=.02  :REM sec between pulses
 1290 time%=&FFFFFFFF -(t*100) +1
 1300 timer%?4=&FF     :REM load highest byte
 1310 !timer%=time%    :REM set up timer, enable events
 1320 PROCstarttimer
 1330 ENDPROC
 1340
 1350 DEF PROCstarttimer
 1360 *FX14,5
 1370 A%=4 :X%=xtimer :Y%=ytimer :CALL &FFF1
 1380 ENDPROC
 1390
 1400 DEF PROCendtimer
 1410 *FX13,5
 1420 ENDPROC
 1430
 1440
 1450 DEF PROCinitial
 1460  REM ****************************************
 1470  REM     Assemble the machine code
 1480  REM ****************************************
 1490 DIM space% 500
 1500 FOR C=0 TO 3 STEP 3
 1510 zeropage=&70   :REM free for users
 1520 portb=&FE60 :osword=&FFF1
 1530 P%=space%
 1540 angle=P% :P%=P%+8  :REM potentially 8 motors
 1550 table=P% :P%=P%+256  :REM 256 possible pulse lengths
 1560 FORI%=table TO table+&100:?I%=&FF:NEXT
 1570 lowtable%=table MOD 256
 1580 hightable%=table DIV 256
 1590 ?zeropage=lowtable% :zeropage?1=hightable%
 1600 [OPT C
 1610 .eventhandler
 1620 PHP:PHA:TYA:PHA:TXA:PHA
 1630 LDA #&04
 1640 LDX #xtimer
 1650 LDY #ytimer
 1660 JSR osword
 1670 LDY #&0
 1680  \Start pulse, for some motors it may be possible to start
 1690  \before filling table and so reduce the wait loop below
 1700 LDA #&FF :STA portb
 1710  \fill table with exceptions
 1720 LDX #&7 :LDA #&FF :CLC \set up bit pattern
 1730 .exceptions
 1740      ROR A :PHA  \bit pattern
 1750      LDY angle,X \get offset corresponding to angle of motor X
 1760      AND (zeropage),Y \keep existing bit pattern
 1770      STA (zeropage),Y \but modified for motor X
 1780      PLA :DEX
 1790      BPL exceptions
 1800 \table is now loaded, fill in some time
 1810 LDY #&60
 1820 .wait DEY
 1830       BNE wait
 1840 LDA #&FF  \all pulses on
 1850 LDY #&0
 1860 .loop AND (zeropage),Y  \but mask off with each table
 1870       STA portb  \element in turn
 1880       INY
 1890       BNE loop
 1900 LDX #&7 :LDA #&FF
 1910 .clear
 1920      LDY angle,X  \clear all the exceptions again
 1930      STA (zeropage),Y
 1940 DEX
 1950 BPL clear
 1960  \all pulses should now be finished
 1970 PLA:TAX:PLA:TAY:PLA:PLP
 1980 RTS
 1990
 2000
 2010 .moveservo \ ....... mch:code smooth move  .......
 2020 .Loop2
 2030 LDX #0
 2040 LDY #0
 2050 .Loop1
 2060   LDA newpos%,X   \  new position for servo x
 2070   CMP angle,X     \  is old position the same?
 2080   BEQ moved       \ if = then do nothing
 2090     BCS add       \ if > then add
 2100        DEC angle,X    \ else take one away
 2110        JMP incrementx
 2120      .add
 2130        INC angle,X
 2140        JMP incrementx
 2150      .moved
 2160        INY
 2170     .incrementx
 2180    INX :CPX #4
 2190    BNE Loop1       \ have i checked all 4 servos?
 2200 JSR waitmotors
 2210 CPY #4
 2220 BNE Loop2          \ continue if not all finished
 2230 RTS                \ got this far then all servos are in correct pos
 2240
 2250
 2260 .waitmotors
 2270 TXA:PHA:TYA:PHA
 2280 LDY #&FF :LDX &81
 2290 .waiting
 2300   DEY :NOP :NOP
 2310   BNE waiting   \ 8 clock cycles loop, ie 1024 cycles all in
 2320 LDY #&FF
 2330 DEX
 2340 BNE waiting
 2350 PLA:TAY:PLA:TAX
 2360 RTS
 2370
 2380 ]
 2390 NEXT
 2400 !&220=eventhandler OR (!&220 AND &FFFF0000)
 2410 ENDPROC
```

STRUCTURAL LINGUISTICS

In PASCAL, we can define our own procedures and functions to supplement those already built into the language. This means, however, that we must fully understand the differences between the two in order to structure our programs efficiently. Here, we look at this aspect, as well as the important concepts of 'scope' and 'parameter passing'.

Before completing our survey of data structures in PASCAL, we'll take a look at some mechanisms for structuring programs. This is done by defining our own procedures and functions to supplement those already built into the PASCAL language, such as the procedure write and the sqrt function. But before defining any of our own, let's review why write is a procedure and not a function, and why sqrt is a function and not a procedure. When we say, for example:

```
write ('Hello!')
```

we expect the character string supplied as the single parameter to appear on the standard output file. In other words, write identifies a subprogram that knows how to send data in the form of a character stream to the output device (VDU). But what would the following 'statement' do?

```
sqrt (256)
```

The answer, of course, is that it would generate a compile-time error message — it isn't a legal statement. On the other hand:

```
write (sqrt (256))
```

would not only be legal, but it should produce something like 1.60000E+01 on the screen. Ask yourself why this is so, and why it does seem so inevitable and obvious — if you can appreciate the reasons, you will not be in any doubt as to the fundamental difference between a procedure and a function.

The identifier write invokes a process that sends data to an output stream and is a procedure that can be called by its name. Another PASCAL procedure is page (F), which is used to start a new page on the text file F. The sqrt identifier, however, must always be supplied with a single numeric 'argument' and it merely returns a result — the real number that is the square root of its argument. So the first distinction between procedures and functions is that there is no such thing as a function statement. Just as writing 16 in isolation means nothing (it's only a value), expressions such as sqrt (X/3) or odd (N) have no meaning on their own. The function identifiers behave rather like variables that are never initialised, but are computed every time their name is used in a program statement. The single value result that is returned is evaluated from an inspection of the current value (or values) of the argument(s) to the function — that is, the result is a function of the argument(s).

Procedures, on the contrary, do not return a value and thus cannot be used in expressions. This again is not surprising. The statement:

```
N: = WriteLn
```

clearly demonstrates its illegality, just as LET N = PRINT would in BASIC.

Writing large programs in PASCAL is made much easier by having the ability to name our own procedures and functions and control their accessibility and the linkage between them. Just as with the naming of variables, the relative lack of restrictions on identifiers means that we can choose helpful and meaningful names for our own subprograms. Consider the following problem:

```
PROGRAM Incomplete (input, output, datafile);
                        {declarations . .}
BEGIN
   Open (datafile);
   WHILE NOT EoF (datafile) DO
      BEGIN
         read (datafile, item);
         Process (item)
      END
END.
```

Even though we have not yet dealt with file handling, you should have no difficulty in understanding what the program does. In the declaration part of the program, we would declare one procedure to locate a file of data on backing storage and open it for reading (Open), and another to manipulate each item of data appropriately (Process). The program also uses two of PASCAL's predefined identifiers. The procedure read (which we have used thus far to read textual input) can be used as shown to read any data, structured or not, from a file of the appropriate type. The Boolean function EoF (End of File) returns the value true when the last record of the file supplied as its argument has been read. Again, by omitting the file identifier, this defaults to the standard file input. For text files only, the function EoLn (End of Line) returns true when the last character on a line has been read. Let's see how this applies when defining our own subprograms.

In essence, there is little difference between programs, procedures and functions — they are all modules that may have their own local data descriptions and bodies of statements. The only

change in the syntax is for the heading. A program heading, as we have seen, consists of the reserved word PROGRAM, a user identifier naming the program and a parameter list identifying the files with which the program will communicate. The heading of a procedure substitutes the reversed word PROCEDURE and extends the 'formal parameter list' to include the type identifiers of each data parameter. So, for example:

```
PROCEDURE Lines (NumLines : integer);
  VAR
    N : integer;
  FOR N := 1 TO NumLines DO
      WriteLn
END;
```

The procedure's END is followed by a semicolon, not a full stop as at the end of a program. NumLines, the formal parameter identifier, is passed the actual value in the calling statement:

```
Lines (5)
```

This admittedly trivial procedure is nonetheless useful if we frequently wish to leave several blank lines (five in this case) for clarity of output, and saves us (and the compiler) from having rather boring sequences of WriteLn statements separated by semicolons. This general purpose procedure will allow you to leave as many blank lines as you need, so that Lines(10) will produce 10 blank lines, Lines(20) gives 20 lines, and so on.

SCOPE

This example is also a good illustration of PASCAL's security. The FOR loop control variable *must* always be declared as a true local variable. We could not have said, for instance:

```
For NumLines := 1 TO NumLines DO . . .
```

Although this would be acceptable in a main program, the loop security cannot be guaranteed if a loop controller is non-local, or 'relatively global'. (For more information on 'local' and 'global' variables, refer to pages 593 and 688.) Have you ever spent hours debugging a BASIC program that contained a statement like:

```
300 FOR N = 1 TO T
400 GOSUB 2000
500 NEXT N
```

only to find that a remote subroutine, perhaps called indirectly and conditionally, used N for some other purpose? PASCAL expressly forbids such uncontrolled behaviour, and helps save wasted development time. Any identifier declared within a block has a defined 'region' that extends throughout that block. However, its 'scope' (that part of the block from within which it is accessible) ony extends from the point of its declaration to the end of that block, and this may be limited further by redefinition. In the previous example, the N referred to within the procedure Lines is, naturally, the locally declared integer, and this declaration temporarily overrides any relatively global one throughout each activation of the procedure.

Referring to the main program outline (Scope) in the example, the region of the global variables N and X is the whole of the program. The scope of X is from its declaration to the end of the program excluding anywhere within procedure A. This is because another X (SET OF char) is defined as a type identifier and its region is the block of A. Similarly, within B, N refers to B's formal parameter, not the global variable.

Although the region of both A and B is the entire program, the scope of B only commences at its defining point, and so while you could use A in procedure B, B is invisible to A. This reinforces PASCAL's trenchant logic: no program can be executed until it has been written! This also explains why the definitions and declarations *must* be in the order CONST, TYPE, VAR, followed by procedure and function definitions ordered according to the structural requirements of the program. Many PASCAL compilers are 'one-pass' (they read the source code only once), and this would not be possible if this logical ordering were not insisted upon.

The extra effort of having to declare data local to any procedure is amply repaid. Modularity is achieved by the passing of all data values as parameters to procedures, and although you may see PASCAL programs that use procedures without any parameter lists, accessing all data globally, this practice should be strictly avoided. In fact, it can be seen as one of PASCAL's weaknesses insofar as it will permit this sort of abuse. Notice that, just as any reference to X within A means the type X, using N within B refers to its local formal parameter and the global integer N is temporarily inaccessible. Apart from the inherent security, this enables a team of programmers to work on a large project without having to worry about identifier clashes.

A Lot Of Scope

```
PROGRAM Scope;

VAR
      N    : integer;
      X    : real;
PROCEDURE A ( Y : real);

   TYPE
      X = SET OF char;
   {etc.}

PROCEDURE B ( N: integer );
   {etc.}

BEGIN {Main program —
   Scope}
   . . .
   A (succ (N)/ 3);
   B (N);
   A (X)
      {etc.}
END.
```

In the program Scope, outlined here, the regions and scopes of the different variables and procedures are shown in the following table:

Proc/Var	Region	Scope
A	main	A,B,main
B	main	B,main
N (global)	main	A,main
X (real)	main	B,main
X (in proc A)	A	A
N (in proc B)	B	B

The call to B passes the value of the global N as an actual parameter, which happens to be called the same name in B but is an entirely separate variable. This implies several important points:

- A local copy of the passed value is made on entry to a block.
- Any change to this 'value parameter' within its block does not affect the actual parameter passed from the calling point.
- The value passed may be any expression of the correct type.

Naturally, any procedure statement must list the actual parameters of the call, and they must match the formal parameter list in the heading in number, position and type. The mechanism for parameter value passing may be envisaged as initialising the formal parameter identifier to whatever value is specified in the calling statement. Thus:

```
Lines (succ(N+gap)DIV 2)
```

implies the following assignment statement upon entry to Lines:

```
NumLines := succ (N+gap) DIV 2
```

Bearing in mind this secure way of passing values, you might like to figure out how we can possibly write a procedure to process some data when it is required to alter that data's values.

From Base To Base

The program BaseValue illustrates the use of procedures with value parameters. Any base from 2 (binary) to 16 (hexadecimal) may be selected, and decimal numbers entered from the keyboard will be displayed in the appropriate notation. For example, 32767 — MaxInt on several small PASCAL compilers — would become 7FFF in hexadecimal, 111111111111111 binary or 77777 in octal.

Try implementing the following modifications:

- To cope with negative representation, we must convert the number to its two's complement. At machine level, this is done by negating it and adding one. Can you think of an easy way to do this in PASCAL?
- Maybe you would like to perform conversions the other way round? If so, you could use the program as a model for one to take a number in any base (2 to 16) and to output the result in decimal form. Perhaps you could incorporate this as a procedure in the program above?

A clue: think about the data and linkage

```
PROGRAM         BaseValue    ( input, output );
        { converts decimal numbers to ANY base
          in the range binary to hexadecimal }
CONST
        Columns  = 79;  { per screen/printer }

TYPE
        byte     = 0 .. 255;
        cardinal = 0 .. MaxInt;

VAR
        number,
        base     : integer;
        FedUp,
        legal    : boolean;

        {11111111111111111111111111111111}

PROCEDURE   WriteDigit  ( digit : byte );
    { value of list [ count ] passed to digit }
  BEGIN
    IF digit IN [ 0 .. 9 ]
      THEN
        write ( digit : 1 )
      ELSE      { represent as A .. F }
        CASE digit OF
          10    : write ( 'A' );
          11    : write ( 'B' );
          12    : write ( 'C' );
          13    : write ( 'D' );
          14    : write ( 'E' );
          15    : write ( 'F' )
        END     { CASE }

  END;  { WriteDigit }
        {11111111111111111111111111111111}

PROCEDURE     Print   ( N       : cardinal;
                        base    : byte );
  CONST { all this data is local to Print }
        MaxDigits = 32;
  TYPE
        bounds    = 1 .. MaxDigits;
  VAR
        list    : ARRAY [ bounds ] OF byte;
        index   : bounds;
        count   : byte;
  BEGIN
    count := 0;

    REPEAT
        count := succ ( count );
        list [ count ] := N MOD base;
        N := N DIV base

    UNTIL N = 0;
                { print starting with MSB : }
    FOR count := count DOWNTO 1 DO
        WriteDigit ( list [ count ] )

  END;  { Print }
        {11111111111111111111111111111111}
BEGIN   { BaseValue - Main program }

  REPEAT
    WriteLn ( 'Choose a number base : ' );
    WriteLn ( '    2 .. 16      ' : Columns );
    WriteLn ( '(any other quits)' : Columns );
    write ( 'Base ? ' );
    read ( base );
    legal := base IN [ 2 .. 16 ];

    IF legal THEN
      BEGIN
        write ( 'Number (0 changes base) ? ' );
        read ( number );
        FedUp := number <= 0;

        WHILE NOT FedUp DO
          BEGIN
            write ( number : Columns DIV 2,
              ' to base ', base : 1, ' is ' );
            Print ( number, base );
            WriteLn;
            write ( 'Number ? ' );
            read ( number );
            FedUp := number <= 0;
          END
      END

  UNTIL NOT legal

END .
```

DATABASE

Here, courtesy of Acorn Computers, we complete the list of the call addresses of the BBC Micro's BASIC ROM routines. Note that these addresses are for BASIC II only

Set MAN#1=MAN#2	A4E8
String addition	9C15
String assignment	8C1E
String comparison	9AE7
String concatenation	9C15
Swap FAC#1 and (&4B)	A4D6
TAB(X	8E24
TAB(X) routine	8E40
TAN function routine	A6BE
TIME function routine	AEB4
TIME statement routine	92C9
TOP function routine	AEDC
TRACE statement routine	9295
TRUE function routine	ACC4
Token table	8071
Tokenise a line of text	8951
Tokenise line numbers	88DB
UNTIL statement routine	BBB1
USR function routine	ABD2
Unpack FAC#1 from (&4B)	A3B2
Unpack FAC#2 from (&4B)	A34E
Unstack a parameter	8CC1
VAL function routine	AC2F
VDU statement routine	942F
VPOS function routine	AB76
WIDTH statement routine	B4A0
Warm start	8AF3
Zero FAC#1	A686
Zero FAC#2	A453
^ operator routine	9E35

In issue 61 we begin publishing the first of several instalments, in which we will provide full details of the Commodore 64's memory map. This series is designed to run concurrently with the Machine Code course on the Commodore 64's operating system